

A GRAPHICAL SIMULATOR FOR TWO-WAY
LINEAR-BOUNDED AUTOMATA (LBA)

By

JOJI DOI

Bachelor of Science

Baker University

Baldwin City, Kansas

1997

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May 2001

A GRAPHICAL SIMULATOR FOR TWO-WAY
LINEAR-BOUNDED AUTOMATA (LBA)

Thesis Approved:

Mansur Samadzadeh-H.

Thesis Adviser

D. E. Hedlund

J. Chandler

Alfred R. Meyer

Dean of the Graduate College

PREFACE

An LBA is an accepting device that can be implemented and simulated in a programming language. To simulate the LBA machine mode, a sophisticated simulator/tool must be designed and implemented. The main purpose of providing the simulator/tool is to use it as a pedagogical aid. In order to increase the user-friendliness, a GUI can be incorporated in the tool. The representation of the tool is an exceedingly important issue when the representation is graphical. This thesis was an investigation to design and implement a sophisticated GUI-based simulator/tool for LBA. Java was utilized for the implementation of the tool.

The main objective of this thesis was to design, develop, and implement a graphical simulation tool that can help in developing LBA. This tool can be used to design state diagrams, construct alphabets, and execute LBA. The user-interface components were implemented using Swing technologies (lightweight component) based on look-and-feel ideas. The tool was first tested by using it to develop several different LBA. The tool was evaluated in two stages: first by over 12 graduate students and second by the students in a junior class on theoretical foundations of computing. The tool was improved, retargeted, and tuned after the first stage of evaluation. Overall, the two stages of evaluation indicated that the tool has sufficient flexibility and can be used effectively.

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my graduate advisor, Dr. Mansur H. Samadzadeh, for his guidance, assistance, and kindness in completing my thesis. I would also like to thank my committee members, Drs. George E. Hedrick and John P. Chandler, for their helpful contributions and advice.

Moreover, I wish to express my sincere gratitude to the Computer Science Department for supporting me during my graduate studies.

I would like to extend special appreciation and gratitude to my parents, who always believed in me and my abilities, for their moral and financial support and encouragement. I am also grateful to my friends and colleagues for their invaluable suggestions and support.

TABLE OF CONTENTS

Chapter	Page
I INTRODUCTION	1
II LITERATURE REVIEW	3
2.1 Definition of Deterministic LBA	3
2.2 Definition of Non-deterministic LBA	4
2.3 Transition	5
2.4 Acceptance	7
III IMPLEMENTATION ISSUES	8
3.1 Java 2 SDK and Java Programming Language	8
3.2 Implementation	9
3.2.1 Mathematical Formula for Drawing an Arrowhead	10
3.2.2 Zooming Mechanism	11
3.2.3 Transition Table	12
3.3 Graphical User Interface	12
3.3.1 File Open Window.....	16
3.3.2 Tape Window	17
3.3.3 Other Windows	18
IV RELATED WORK AND PROPOSED TOOL	22
4.1 Related Work	22
4.2 LBA Tool	23
V TESTING AND EVALUATION OF THE TOOL	25
5.1 Testing	25
5.2 Evaluation	27
VI SUMMARY AND FUTURE WORK	29
6.1 Summary	29
6.2 Future Work	30

REFERENCES	32
APPENDICES	34
APPENDIX A – GLOSSARY	35
APPENDIX B – TRADEMARK INFORMATION	36
APPENDIX C – USER GUIDE FOR THE LBA TOOL	37
APPENDIX D – PROGRAMMER GUIDE FOR THE LBA TOOL.....	43
APPENDIX E – EVALUATION FORM	45
APPENDIX F – SELECTED PROGRAM LISTINGS.....	47

LIST OF FIGURES

Figure	Page
1. A non-deterministic LBA	5
2. A deterministic transition	6
3. Structure of the LBA simulator	10
4. Arrowhead Coordinates	11
5. Selection window for the input alphabet.....	14
6. Main window of the LBA simulator	15
7. File dialog box	16
8. Tape window	17
9. State diagram window	19
10. Transition speed control window	20
11. User guide help window	20
12. Example of the tree view of multiple processes for non-deterministic computations	21
13. Transition table window	21
14. Tape string generator window.....	40

CHAPTER I

INTRODUCTION

Turing machine (TM) is the accepting device of a language that is generated by a Type 0 or unrestricted grammar [Martin 97] [Sudkamp 97]. Some of the characteristics of a TM, as a recognition or acceptance device with an infinite length input tape, prevent it from being implemented. LBA (linear bounded automata or automaton) is the accepting device of a language that is generated by a Type 1 or context-sensitive grammar. Technically, an LBA can have a tape of any bounded length but, by using a finite length tape, an LBA can be implemented (simulated) in a programming language.

A simulator was designed and implemented for LBA with a graphical user interface (GUI) which incorporates a subset of Java™ Application Programming Interfaces (APIs) [Lewis and Loftus 00]. The tool consists of set of necessary and convenient functions in order for the user to operate and manipulate an LBA. Generally speaking, the complexity of using a software simulation tool depends on the user interface of the simulator. Thus, GUI design and implementation is exceedingly important. Abstract Windows Toolkit or Abstract Windowing Toolkit (AWT) [Flanagan 97] and Java Swing were utilized to provide an enhanced graphical representation environment for the tool user.

The main objective of this thesis was to develop a graphical tool which can help in simulating an LBA with a given configuration and constructing an LBA. The rest of this thesis is organized as follows. Chapter II of this thesis provides a literature review on LBA starting with the formal definition of LBA. The implementation of the tool, including the graphical representation issues, are discussed in Chapter III. In Chapter IV, related tools and their functionalities are introduced. In Chapter V, the testing and evaluation of the tool are discussed. Finally, the summary of the thesis and future work are presented in Chapter VI.

CHAPTER II

LITERATURE REVIEW

LBA is a machine model that accepts or recognizes context-sensitive languages which are languages generated by context-sensitive or Type I grammars [Sudkamp 97]. The productions of Type I grammars have the form $\alpha \rightarrow \beta$, where $\alpha, \beta \in (V \cup \Sigma)^*$ for V a set of non-terminals and Σ a set of terminals, α must contain at least one non-terminal, and the length of α must be less than or equal to the length of β [Martin 97]. The LBA machine model is constructed to simulate the derivations of context-sensitive grammars [Sudkamp 97]. An LBA can be graphically represented by a transition diagram [Gurari 89], where a transition $\delta(q_i, x) = [q_j, y, d]$, where $d \in \{L, R, S\}$, can be depicted by an edge from q_i to q_j labeled $x / y d$. The formal definition of LBA is given below.

2.1 Definition of Deterministic LBA

Sudkamp and Martin [Martin 97] [Sudkamp 97] formally define LBA as follows. An LBA is a structure $M = (Q, \Sigma, \Gamma, \delta, q_0, <, >, F)$, where Q is a finite set of states, Σ is the input alphabet that is a subset of $\Gamma - \{blank\}$, Γ is the tape alphabet, δ is a partial function that is the transition function from $Q \times \Gamma$ to $Q \times \Gamma \times D$ for a deterministic LBA

with D as a set containing the three directions of left, right, and stationary (L,R,S), $q_0 \in Q$ is the initial state or start state, $<$ and $>$ are the distinguished symbols of Σ , and $F \subseteq Q$ is a set of final states. The machine is not able to recognize any character that comes to the left side of the symbol $<$. The machine is not able to recognize any character that comes to the right side of the symbol $>$.

The above definition of an LBA is for a deterministic machine. A deterministic LBA is a special case of a non-deterministic LBA. Non-deterministic LBA are described below.

2.2 Definition of Non-deterministic LBA

In a deterministic LBA, out of any given configuration there is at most one transition. A non-deterministic LBA may specify any finite number of transitions out of a given configuration [Sudkamp 97]. A non-deterministic LBA can be defined as a deterministic LBA (defined above in Subsection 2.1) with the exception of the transition function. A transition function in a non-deterministic LBA maps $Q \times \Gamma$ to subsets of $Q \times \Gamma \times D$. An example of a non-deterministic LBA is given below.

$M = (Q, \Sigma, \Gamma, \delta, q_0, <, >, F)$ is an LBA, where $Q = \{t_0, t_1, t_2, t_3, t_4\}$ is the set of states, $\Sigma = \{a, <, >\}$ is the input alphabet, $\Gamma = \{a, A, B, <, >\}$ is the tape alphabet, $\delta = \{[(t_0, <), (t_1, <, R)], [(t_1, a), (t_2, A, R)], [(t_1, a), (t_3, B, R)], [(t_2, a), (t_2, A, R)], [(t_3, a), (t_3, B, R)], [(t_2, _), (t_4, _, S)], [(t_3, _), (t_4, _, S)], [(t_2, >), (t_4, >, S)], [(t_3, >), (t_4, >, S)]\}$ is the transition function, q_0 is the initial state, $<$ and $>$ are the end-markers, and $F = \{t_4\}$ is the set of final states. Figure 1 shows a graphical view of the transition diagram of machine M .

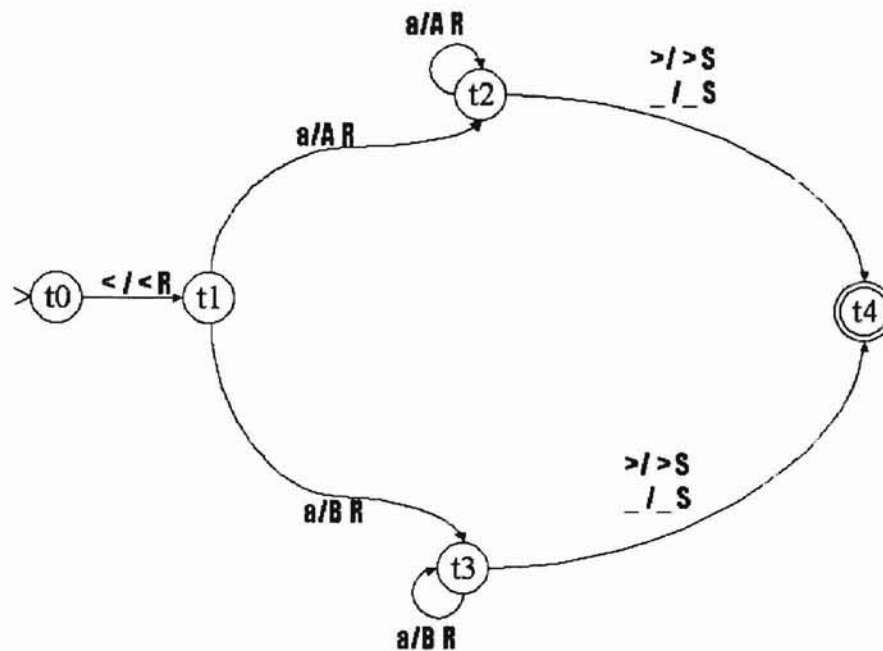


Figure 1. A non-deterministic LBA

In the above transition diagram, the initial state t_0 has an arrowhead as a distinguishing symbol, and the double circle distinguishes the final state t_4 . The machine processes non-deterministically at state t_1 if an a is encountered. When this occurs, the machine produces two computations for an input string.

2.3 Transition

A transition consists of three actions: changing the state, writing a symbol on the square scanned by the read/write head, and moving the read/write head. The direction of the movement is specified by the final component of each transition [Sudkamp 97]. An L indicates a move of one tape position to the left, an R indicates a move of one tape position to the right, and an S indicates no move of the tape position.

An example of a deterministic transition is given in Figure 2. The transition used is $\delta(q_0, <) = [q_1, <, R]$. In this example, the tape of the LBA consists of seven cells. The tape positions are numbered for easy reference. A computation begins with the machine in state q_0 and the read/write head scanning the leftmost position. The input, $aab \in \Sigma^*$, is written on the tape beginning at position one. The positions zero and six of the tape contain the end-markers $<$ and $>$. The remainder of the tape is to be blank. A blank is denoted by an underscore.

The first step of a transition is changing the state from q_0 to q_1 . The second step is replacing $<$ with $<$. As the final step, the read/write head moves to the right because the final component of the transition indicates an R.

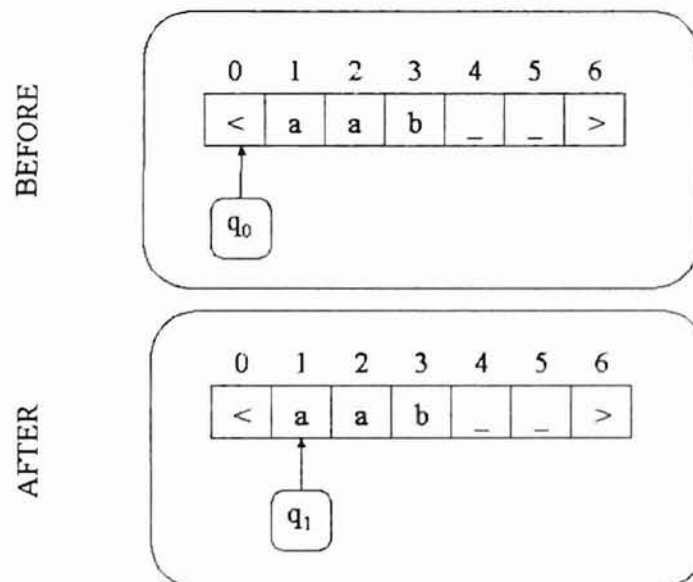


Figure 2. A deterministic transition

2.4 Acceptance

A string $x \in (\Sigma - \{<, >\})^*$ is accepted by an LBA if a computation with input x halts in an accepting state [Sudkamp 97]. It is called normal termination if the machine halts in an accepting state. On the other hand, a machine halts abnormally if one of the following conditions holds: when it encounters a state, symbol pair for which no transition is defined, when it reads $<$ and L , or when it reads $>$ and R [Sudkamp 97].

CHAPTER III

IMPLEMENTATION ISSUES

3.1 Java 2 SDK and Java Programming Language

Java 2 Platform Software Development Kit (Java 2 SDK) is a software development environment developed and distributed by Sun Microsystems, Inc. The latest version of Java 2 SDK Standard Edition, v 1.3, is available for download at the official Sun website (<http://java.sun.com>). Java 2 SDK comes with APIs and executable tools such as `java.exe` and `javac.exe`. They are used when a developer compiles java source code and runs a program. Programs run on the Java Virtual Machine (VM).

Java is an object oriented programming language and an interpreted language which can be compiled and executed on Java 2 SDK. Java is a portable and platform-independent or architecture-neutral language, because the Java bytecode format is not associated with any particular hardware platform. A java application can run on any system as long as that system implements the VM [Flanagan 97] [Lewis and Loftus 00]. The Java compiler generates Java bytecode from Java source code, and the Java interpreter translates and executes the bytecode [Lewis and Loftus 00].

3.2 Implementation

The software for the LBA simulator is divided into several classes organized as objects. Some of these classes are from API packages and are loaded at the execution time. A class is a collection of data members and methods, and methods manipulate data members [Flanagan 97]. The tool has been designed and implemented as an event-driven application. An event-driven application is defined as a program that detects and acts upon a user action or a system occurrence [AOL 00]. The tool consists of Main Routine, User Interfaces, Event Listeners, and Event Handlers. After the tool is invoked by the user, Main Routine invokes the GUI components to initialize the user interface. After the initialization of GUI, Event Listeners sleeps and listens to the occurrence of an action/event which is triggered by the user. Upon any occurrence of actions/events, Event Listeners reacts and invokes appropriate Event Handlers. Figure 3 shows the structure of the tool. Event Handlers will invoke four major routines that are File I/O, Print Job, Create LBA, and Run LBA.

File I/O routine performs two tasks. One task is to save the LBA configuration and the tape string information created by the user in binary data format. The binary data is stored in a user specified file with an lba extension. The other task is to open a file that has a specific format with the file name having an lba or a tpe extension. The specific format can be generated by a feature of the tool, i.e., by the save option under the File menu. If the data file has incorrect format or is corrupted, the tool informs the user via a popup message box. The Print Job routine creates printable image of the current state diagram and sends the image as a printer job to the user-specified printer. The Create

LBA routine will be invoked when the user selects New under the File menu. It enables the user to construct the input alphabet, the tape alphabet, and the state diagram. The Run LBA routine is invoked when the user selects RUN. In the RUN mode, the user can simulate the LBA and view an animation of the state diagram.

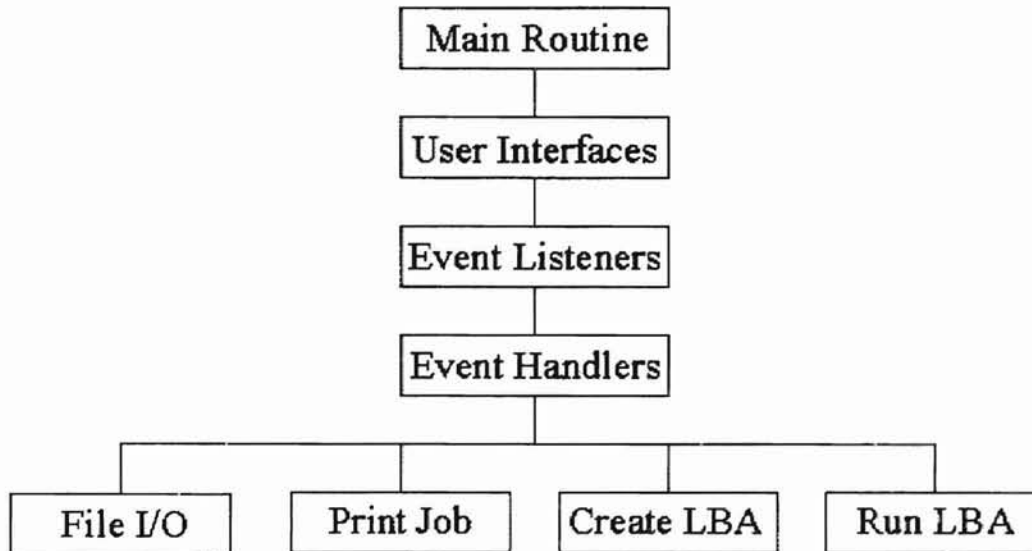


Figure 3. Structure of the LBA simulator

3.2.1 Mathematical Formula for Drawing an Arrowhead

The LBA tool allows the user not only to construct and edit, but also to move objects that have already been created by the user. The arrow that represents a transition between two states is drawn automatically. The location of the arrow is calculated based on the location of the states and the labels. The following mathematical formula [Hassan 93] was used in order to find the coordinates for an arrowhead (Figure 4) from the given coordinates:

$$Cx = Bx + (length * \cos(\alpha + \beta))$$

$$Cy = By + (length * \sin(\alpha + \beta))$$

$$Dx = Bx + (length * \cos(\alpha - \beta))$$

$$Dy = By + (length * \sin(\alpha - \beta))$$

where $\beta = 20.0 * 0.01745329$ and $\alpha = \tan^{-1}((By - Ay)/(Bx - Ax))$.

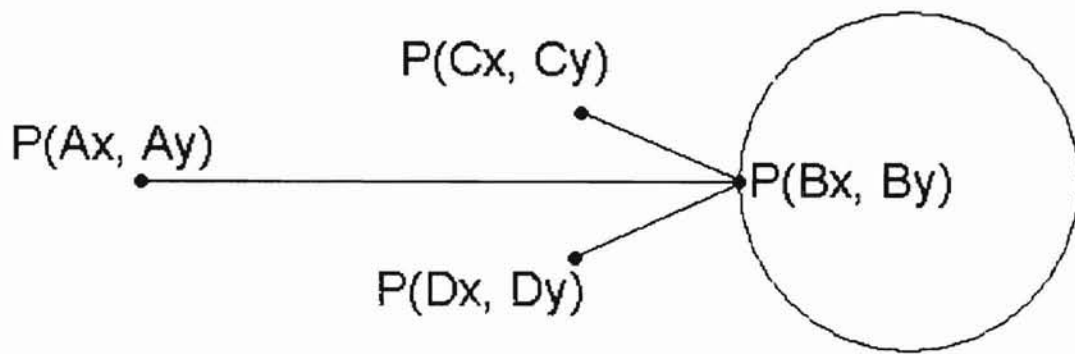


Figure 4. Arrowhead Coordinates

3.2.2 Zooming Mechanism

The tool has a zooming mechanism on the state diagram that is in its window. When the tool receives the zooming request from the user, it checks whether it is within the zoom boundary. If the request is not beyond the boundary, the tool will compute new coordinates of every state in the window. For this computation, the following formulas are used:

For zoom in:

$$\text{NewLocationX} = \text{OriginalStateX} + \text{DistanceWidth} / 2$$

$$\text{NewLocationY} = \text{OriginalStateY} + \text{DistanceHeight} / 2$$

For zoom out:

$$\text{NewLocationX} = \text{OriginalStateX} - \text{DistanceWidth} / 3$$

$$\text{NewLocationY} = \text{OriginalStateY} - \text{DistanceHeight} / 3$$

where distance can be obtained from the following formulas.

$$\text{DistanceWidth} = \text{OriginalStateX} - \text{CenterFrameX}$$

$$\text{DistanceHeight} = \text{OriginalStateY} - \text{CenterFrameY}$$

3.2.3 Transition Table

The implementation of the transition table in Java is difficult because Java API does not have a flexible way to construct a table. The transition table was implemented with the default table model which is available in Java API; however, the default does not include a row header that should show the states. Thus, it was required to integrate the table with the List object. The List object displays the states. The table used in the tool was designed and implemented based on the idea created by Nobuo Tamemasa [Tamemasa 00].

3.3 Graphical User Interface

The user-interface of the simulator was developed utilizing some of the Swing components. Swing components are the software packages for graphical user interface

that come with Java SDK v1.2. Utilizing Swing for GUI reduces system overhead [Topley 98]. Most of the Swing components are derived from lightweight component of `java.awt.Component` class. Lightweight component does not depend on the native windowing system [Topley 98].

The GUI for the tool was designed based on the guidelines imparted by Lewis and Loftus [Lewis and Loftus 00]. The guidelines are that the interface must address user needs, limit keyboard input, and prevent user errors and must provide mnemonic short cuts whenever reasonable as well as consistent user interface throughout the tool.

The GUI for the LBA tool facilitates a zoom in/out mechanism for the state diagram and an undo function for the transition mapping, as well as an editing capability for the input alphabet, the tape alphabet, and the tape string after their creation. These functionalities address the needs of the tool user. The selectable input value reduces the possibility of introducing typing errors. The tool responds with a popup window (Figure 5) to a user's create/edit input/tape alphabet. The popup window consists of the Ok button and a set of check boxes labeled with symbols. Mnemonic short cuts are available for STEP, UNDO, GO, STOP, ZOOM IN, and ZOOM OUT. For example, the user can zoom in by holding the Alt key and pressing the I key. In order to keep the tool consistent, Look-and-Feel design for the menu bar and menu items as well as the icons was integrated.

The main user interface of the tool is shown in Figure 6. The main window contains three buttons on the upper right corner, a menu bar, six icons, two radio buttons, and the edit option selection box. The buttons on the upper right corner from left to right

can be used to minimize, maximize, and close the window, respectively. In order to keep the original size of the window, the maximize button and the resizing option are disabled.

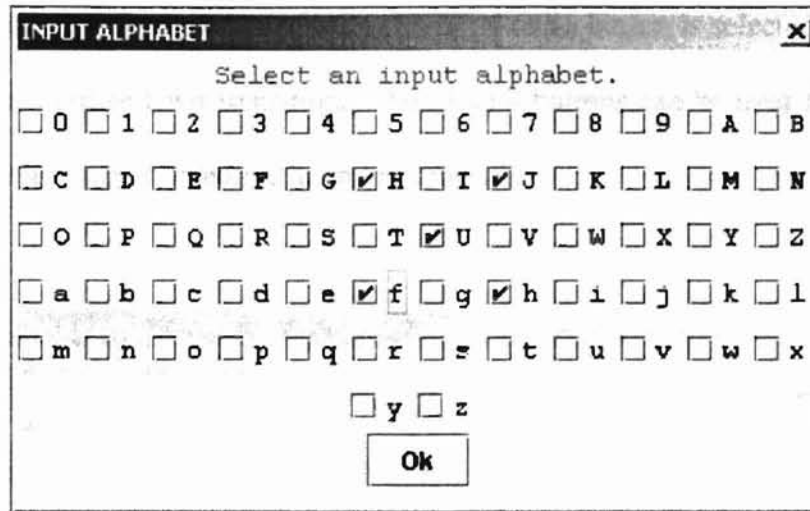


Figure 5. Selection window for the input alphabet

There are four menus in the menu bar: File, Configure, View, and Help. The File menu contains menu items that are file operations such as loading an LBA machine configuration and an input tape string from files, saving the current position of the state diagram into a file, and resetting the current machine configuration and the input tape. The simulator will be terminated and the executing program will be halted when the Exit menu item under File is clicked. The Configure menu contains items that construct/edit the input alphabet, the tape alphabet, and the tape string. The View menu contains items that show/hide other windows, such as the transition speed control window, the transition function window, the tree view of the multiprocess window, and the tape string window. The Help menu provides the user guide and the online source code documentation. The

online source code documentation includes all public and non public classes, methods, and instance variables. The source code of the tool is not available online.

Icons are located below the menu bar. First four icons from the left are disabled in the Edit mode. They are enabled after the RUN radio button is selected, because they are used to manipulate LBA transitions. The Zoom buttons can be used for zooming in and out the state diagram in both edit and run modes.

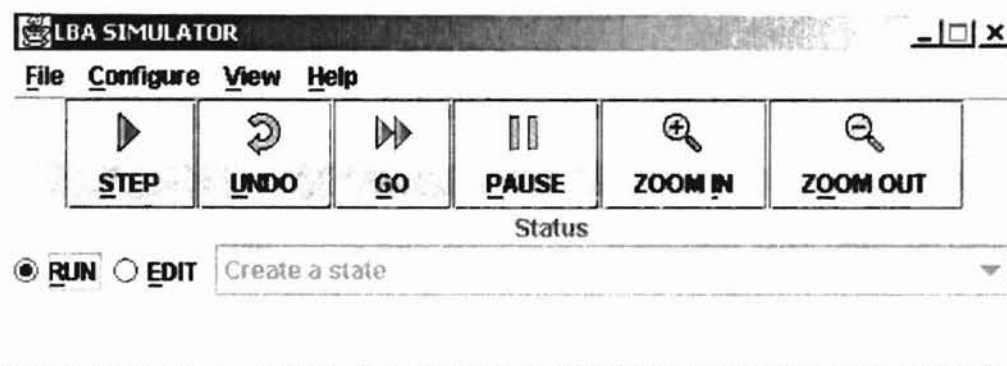


Figure 6. Main window of the LBA simulator

Two radio buttons under the STEP icon switch between the edit mode and the run mode. In the edit mode, the user can create/change the state diagram; while in the run mode, the user can simulate the machine and animate the state diagram.

The edit option selection box, which is located to the right of the EDIT button, contains a set of operations for editing the state diagram. These operations are: create a state, create a transition, move a state / transition, move selected area, delete a state, delete a transition, select the start state, select the final state, and edit a transition. The details of these operation are available in Appendix C.

3.3.1 File Open Window

The user can construct an LBA and a tape string by retrieving an existing binary file. An LBA configuration file has an lba file extension. A tape string file has a tpe file extension. Both operations can be effected by clicking the menu items under File. Clicking the Open File menu item under the File menu causes a popup file dialog box to appear on the screen. In the example in Figure 7, the dialog box lists only files that have an lba extension.

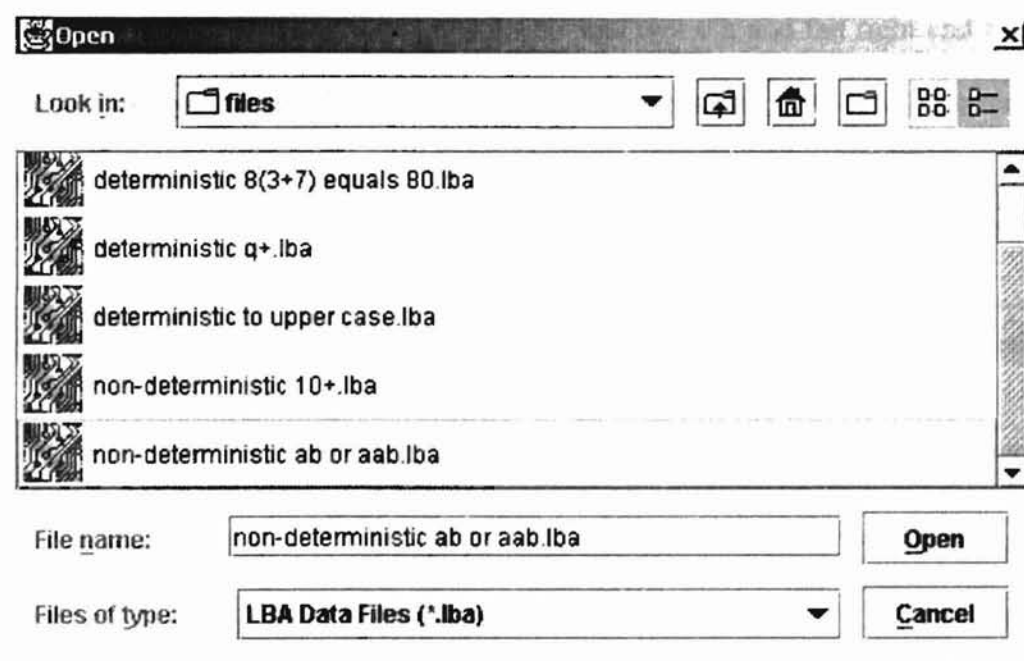


Figure 7. File dialog box

The user can either click on the file name and then click on the Open button or simply double click on the file name to load the data into memory. More description of the windows is given in the following sections.

3.3.2 Tape Window

A tape window (Figure 8) pops up after an input string is constructed or loaded from a file. The tape window displays the string in three segments. The area above the left scrollbar is the first segment (which is empty in the example in Figure 8) depicting a substring of the input string from the leftmost character to the one before the character under the read/write head [Sudkamp 97]. The second segment holds a character that is displayed in the area above the button in the middle of the window. This character is being processed and is called the character under the read/write head. The area above the right scrollbar is the third segment (which contains two 0's and the right end marker in the example in Figure 8) holding a substring of the input string from the character next to the one under the read/write head to the rightmost character.

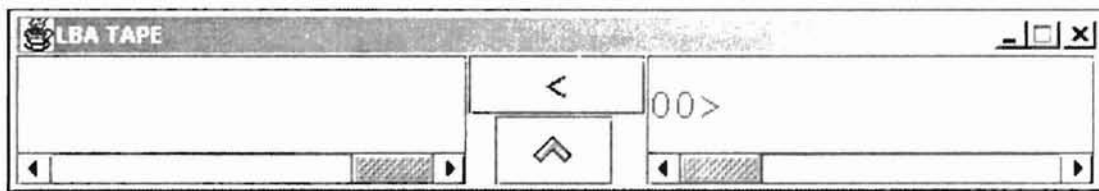


Figure 8. Tape window

Scrollbars are used to view long substrings that do not fit in the display areas. The left scrollbar controls the substring in the segment on the left-hand side, and the right scrollbar scrolls the substring in the segment on the right-hand side. The button in the middle labeled as ^ can be pressed to restore the original position. For instance, assume that the input string is HHHHH LLLL C RRRR hhhhh, where the H's are the hidden characters and the L's are the displayed characters in the first segment or the left-

hand side of the tape window. The character C represents the character under the read/write head in the second segment. The R's are the visible characters and the h's are the hidden or invisible characters in the third segment or the right-hand side of the tape window. After the left scrollbar is moved two units to the left, and the right scrollbar is moved one unit to the right, the following sequence will be obtained: HHH LLLL HH C h RRRR hhhh. Pressing the button that is located between the scrollbars will reset the scrolled sequence of characters or the string back to the original format.

3.3.3 Other Windows

There are some more windows for the tool. Figure 9 shows the state diagram window. The state diagram window enables the user to construct an LBA in the edit mode. In the run mode, the user is able to simulate the LBA with the user defined tape string.

Figure 10 depicts the transition speed control window. This window has a slider to adjust the transition speed at the Go transition mode.

The optional user guide help window is shown in Figure 11. It has two display areas with a vertical scrollbar on each area. The top display area lists the key word(s) of the operation of the tool. The bottom area of the window (the description area) shows the explanation or guide to the operation that the user selects from the top display area. In the example in Figure 11, the user has clicked on the Step and Go key words; therefore, the description area explains to the user how Step and Go functions work. The list of key

words changes dynamically based on the user's action. Also, the user can hide/show this window by clicking a menu item under the Help menu.

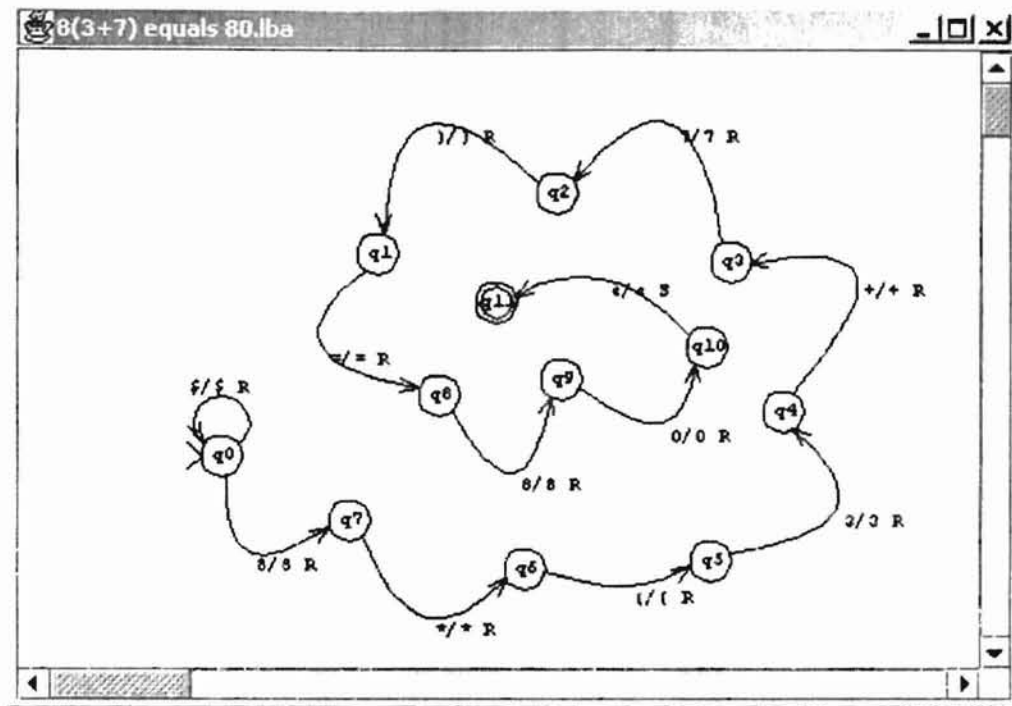


Figure 9. State diagram window

Under a non-deterministic LBA simulation, the user will have more than one state diagram window based on the number of possible transitions. In order to switch among these windows, the user can click the window that (s)he wishes to activate, or the user can utilize the multiprocess view window (Figure 12). This window provides the user tree view of the current number of state diagram windows dynamically. Clicking on a process id in the tree activates the corresponding window. In the example in Figure 12, there are three processes under the root and each process represents a state diagram window.

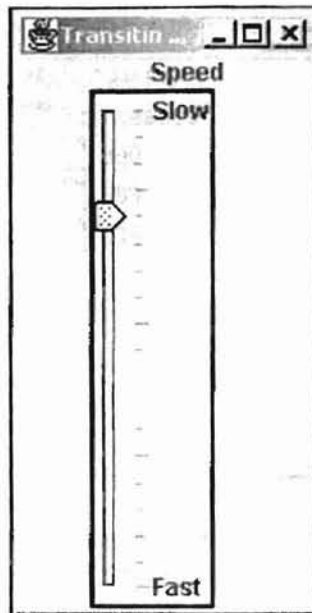


Figure 10. Transition speed control window

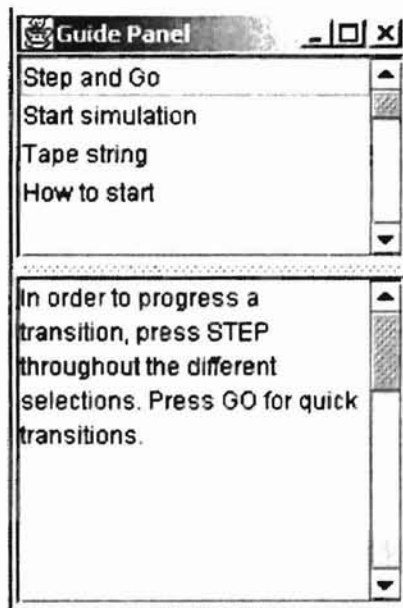


Figure 11. User guide help window

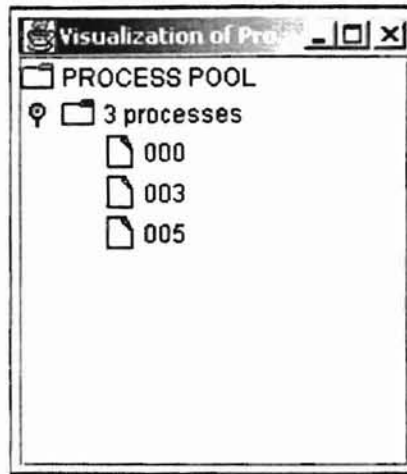


Figure 12. Example of the tree view of multiple processes for non-deterministic computations

The transition table window (Figure 13) is the window that contains the tabular view of the transition function. The row index represents the state names and the column index represents the input alphabet. The matrix contains ranges of the functions. For instance, function $(q_6, 0)$ maps to (q_7, t, R) . Blank cells in the matrix represent the fact that the function is not defined at these points.

Transition Table				
	{	}	0	
q0				
q1				
q2				
q3				
q6				q7, t, R
q7	q7, {, R			q3, t, R
q8				q7, z, R

Figure 13. Transition table window

CHAPTER IV

RELATED WORK AND THE LBA TOOL

4.1 Related Work

There are a number of related simulation tools with GUI. These related tools provide unique representations and features. Two Finite State Automata (FSA) simulators were implemented by Ross [Ross 99] and Faenov [Faenov 96]. Also, an LBA simulator was designed and implemented by Savoiu [Savoiu 00].

The FSA simulators are Java applet programs that run on a web browser. Common among these simulators are features of them are to add a state, to move a state, to delete a state, to add a transition, to move a transition, to delete a transition, to mark an initial state, to mark a final state, to specify an input string, to run an FSA, and to stop the execution of an FSA. Faenov added file input/output functions to his simulator and restricted the maximum number of states to ten [Faenov 96]. Ross [Ross 99] added functions to select stepping/non-stepping modes for the execution, to suspend the execution of an FSA, and to modify a state/transition.

Savoiu's LBA simulator [Savoiu 00] comes in an executable file that runs on Microsoft Windows™ platforms. Major features of the tool are to add a state, to move a state, to delete a state, to add a transition, to move a transition, to delete a transition, to

specify an input string, to specify an input alphabet, to specify a tape alphabet, to run an LBA, and to stop the execution of an LBA.

The above three simulators are well-designed tools in terms of the sophisticated functionalities they provide as well as the clear graphical representation they have. Three features, which are in none of the above simulators, are: zooming control over the state diagram, multiple windows under a non-deterministic computation, and undo operation for the execution of LBA. These features were implemented and incorporated in the LBA tool that was designed and implemented in this thesis work.

4.2 LBA Tool

Design, implementation, testing, and evaluation of a tool for LBA in Java were pursued in the thesis work. The tool was developed with the idea of incorporating user-friendliness. The tool provides the user with a graphical view of a simulated LBA, and it was implemented by adopting event-driven approach. Employing the technology of JFC, such as AWT and Swing, engages the tool with GUI, so that the user can simulate an LBA by simply using the mouse. Furthermore, the tool makes available a help option that gives choices to the user as needed when using the tool. The tool has capabilities to guide the user step-by-step in utilizing the tool.

The LBA simulator is a tool that provides the variety of functions. The functions are divided into two categories. One category contains functions that are used to construct an LBA, for example creating new LBA from scratch or loading a machine configuration and an input string from files. The other category includes functions that

are used to operate and/or manipulate an LBA, for instance zoom in, zoom out, drag, scroll, step-by-step transition, go/stop transition, and undo transition.

CHAPTER V

TESTING AND EVALUATION OF THE TOOL

5.1 Testing

The tool was thoroughly tested with respect to its functionality. Several deterministic and non-deterministic LBA were constructed and simulated successfully using the tool. The testing process aimed to catch implementation errors and to capture possible unexpected behavior of the tool. The following criteria were observed and adhered to through the testing process.

- **Boundary Check:** The tool should inform the user when he/she exceeds certain limits. (i.e., the maximum number of states is 100, the maximum number processes is 50, the maximum number of transitions is 10,000, and the maximum length of the tape is 1,000 including the end-markers). The tool user cannot change these constraints, however the programmer can change these constraints based on the run time support environment, such as capacity of memory and CPU.
- **Transition Speed Range:** The tool should provide for the user a reasonable transition speed range. The speed is determined by the value of the delay between transitions.

The maximum delay (slowest speed) is 1,000 milliseconds or 1 second. The minimum delay (fastest speed) is 100 milliseconds or 0.1 second.

- File I/O: The tool should regulate the file I/O process precisely. It is important that the LBA configuration data and the tape string data be stored into files and be retrieved from the files without having a data change or corruption.
- User Guide Coordination: The tool should guide the user appropriately. User actions should trigger the user guide to add the necessary key words and their descriptions in the user guide window.
- State Diagram Construction: The tool should respond to the user correctly. It is necessary to ensure that each edit operation (i.e., create a state, create a transition, move a state / transition, move selected area, delete a state, delete a transition, select the start state, select the final state, edit a transition) behaves appropriately.
- Window Hide/Show: The tool should be capable of hiding/showing all windows, with the exception of the main window, at the user's request. All windows can be minimized and some windows can be resized.
- Tool Termination: The tool should supply the user an option to exit the tool at any time. Graceful termination of the tool is required. The user is able to terminate the tool at any time by clicking the x button on the main window or by clicking on the Exit menu item under the File menu.

5.2 Evaluation

The evaluation of the tool was conducted in two stages. The first stage of the evaluation was carried out by soliciting the help of over 12 graduate students in the Computer Science Department of Oklahoma State University, Stillwater, Oklahoma. Approximately 10 of the graduate students returned some feedback. The evaluation process included the following: Go to the web site (www.geocities.com/anonbeat/index.html), read the five-step instruction, download the zipped file, unzip the zipped file, execute the lba.jar file, construct your own LBA, simulate the LBA, print out the state diagram, terminate the tool, fill out the online evaluation form (consisting ten multiple choice questions and one optional comment area), and submit it by clicking the submission button on the webpage.

From the feedback obtained from the users of the tool, a number of modifications were made to the original implementation. One of the significant modifications was enabling the user to edit the input alphabet, the tape alphabet, the end-markers, the tape string, and the transition labels. Initially, the tool gave the user only one chance at the creation of these objects, and the user had to delete the original object in order to place a new object, if he/she needed to change an object. For example, creation of an entire LBA from scratch was required when the user needed to change the input alphabet or the tape alphabet. Other improvements were drawing the initial/final state and the transitions automatically when the user is constructing a new state diagram, and an enhancement of the file I/O scheme regarding the input tape. The old version of the tool did not have the

functionality to store the tape string, thus the tape string had to be constructed for every run of the LBA simulator.

The tool was improved, retargeted, and tuned after the first stage of evaluation. Over 50 students participated in the second stage of evaluation of the LBA tool. In the second stage of evaluation, the tool was evaluated by the students in a junior class on theoretical foundations of computing.

A summary of the online evaluation is given in below.

Majority of the students used a Windows 98 environment to run the tool. About 66% of the graduate students felt that using the tool was easy while 50% of the junior students felt that using the tool was easy. Some students reported the following comments.

- The computer became very slow.
- It is difficult to close the various windows.
- The computer froze.
- The screen refreshing was not done properly.
- The computer crashed when it was printing.

Overall, the two stages of evaluation indicated that the tool has sufficient flexibility and can be used effectively.

CHAPTER VI

SUMMARY AND FUTURE WORK

6.1 Summary

In Chapter I, the main objective of this thesis was stated. Chapter II of this thesis provided an introduction to LBA. Sections 1 and 2 of this chapter presented the formal definition of deterministic LBA and non-deterministic LBA. Sections 3 and 4 of Chapter II covered transition rules and acceptance conditions of LBA. The implementation of the tool, including the graphical representation issues, were discussed in Chapter III. Section 1 of Chapter III addressed the implementation platform and the run-time environment including an introduction to the Java 2 SDK and the issue of platform independent. Section 2 of Chapter III provided the program structure of the LBA simulation tool. The last section of this chapter discussed GUI design issues and GUI guidelines. In Chapter IV, a number of related tools and their functionalities were introduced as well as some improvements for the LBA tool. In Chapter V, the test and evaluation of the tool were discussed.

The main objective of this thesis was to design, develop, implement, test, and evaluate a graphical simulation tool that can help in developing LBA. This tool can be

used to design state diagrams, construct alphabets, and execute LBA. The user-interface components were implemented using Swing technologies (lightweight components) based on look-and-feel ideas [Topley 98].

6.2 Future Work

The future versions of this tool should incorporate one or more of the improvements mentioned below.

- A speech functionality can be implemented, so that a blind user can simulate and visualize LBA tool throughout his/her simulation, i.e. designing of LBA, constructing and editing of a state diagram, and animating of the state diagram and tape.
- A voice recognition input system can be implemented in addition to the generic input methods (keyboard/mouse). This will give a user keyboard/mouse free simulation environment.
- A enhancement of the zooming function can be provided by having partial zooming in/out a selected area.
- More efficient algorithms can be designed and implemented to achieve better execution time and better memory usage.
- Sophisticated animation of the state diagram and the tape string can be implemented.
- Enhanced printing options such as printing the tape, transition table, and the tree view of multiple processes can be provided.
- A comprehensive user guide, sample LBA configuration files, and tape string files can be accessible online.

- Finally, as it was originally intended, the LBA tool can be accessible on the Internet through a browser.

REFERENCES

- [AOL 00] AOL, <http://aol.pcwebopedia.com/TERM/e/event.html>, creation date: Not Available, date accessed: September 10, 2000.
- [Faenov 96] Kyril Faenov, www.cs.montana.edu/webworks/projects/fsa-old/fsa.html, creation date: October 21, 1996, date accessed: May 25, 2000.
- [Flanagan 97] David Flanagan, *Java in a Nutshell*, 2nd ed., O'Reilly & Associates, Inc., Sebastopol, CA, 1997.
- [Gurari 89] Eitan M. Gurari, *An Introduction to the Theory of Computation*, Computer Science Press, Rockville, MD, 1989.
- [Hassan 93] Muhammad T. Hassan, "Towards a Graphical Petri Net Tool", Master of Science Thesis, Computer Science Department, Oklahoma State University, Stillwater, OK, July 1993.
- [Lemay and Perkins 97] Laura Lemay and Charles L. Perkins, *Teach Yourself Java 1.1 in 21 Days*, 2nd ed., Sams.net Publishing, Indianapolis, IN, 1997.
- [Lewis and Loftus 00] John Lewis and William Loftus, *Java Software Solutions Foundations of Program Design*, 2nd ed., Addison-Wesley Longman, Inc., Berkeley, CA, 2000.
- [Martin 97] John C. Martin, *Introduction to Languages and the Theory of Computation*, 2nd ed., McGraw-Hill, Inc., San Francisco, CA, 1997.
- [Matsumura and Tayama 86] Kazuo Matsumura and Shuichi Tayama, "Visual Man-Machine Interface For Program Design and Production", *Proceedings of the 1986 IEEE Computer Society Workshop on Visual Languages*, pp. 71-80, Dallas, TX, June 1986.
- [Ross 99] Rocky Ross, www.csm.astate.edu/states/Simulator.html, creation date: March 20, 1999, date accessed: May 23, 2000.
- [Savoiu 00] Nicolae Savoiu, www.ics.uci.edu/~savoiu, creation date: March 20, 2000, date accessed: May 25, 2000.
- [Sudkamp 97] Thomas A. Sudkamp, *Languages and Machines: An Introduction to the*

Theory of Computer Science, 2nd ed., Addison Wesley Longman, Inc., Menlo Park, CA, 1997.

[Tamemasa 00] Nobuo Tamemasa, <http://www2.gol.com/users/tame/>, creation date: November 22, 1998, date accessed: September 14, 2000.

[Topley 98] Tim Topley, *Core Java Foundation Classes*, Prentice Hall PTR, Upper Saddle River, NJ, 1998.

[Tsuda et al. 89] Kazuyuki Tsuda, Masahito Hirakawa, Minoru Tanaka, and Tadao Ichikawa, "An Iconic Retrieval System for Object-Oriented Databases", *Proceedings of the 1989 IEEE Workshop on Visual Languages*, pp. 130-137, Rome, Italy, October 1989.

APPENDICES

APPENDIX A

GLOSSARY

API	Application Programming Interface is a set of tools and program sections that perform particular tasks for building software applications.
AWT	Abstract Windowing (Windows) Toolkit enables programmers to develop Java applications with GUI components, such as windows, buttons, and scrollbars.
GUI	Graphical User Interface.
Java 2 SDK	Java 2 Software Development Kit is a software package contains tools, runtimes, and APIs for developers writing, developing, and running applets and applications in the Java programming language.
JFC	Java Foundation Classes set is software that extends the AWT (see above) by adding a comprehensive set of GUI class libraries.
LBA	Linear Bounded Automata.
Look-and-Feel	The generic form and function of a user interface.
Swing	The software packages for graphical user interface that come with Java SDK v1.2.
TM	Turing Machine.
Type 0 Grammar	A grammar in which all productions are of the form $\alpha \rightarrow \beta$, where $\alpha, \beta \in (V \cup \Sigma)^*$, V is a set of non-terminals and Σ is a set of terminals. α must contains at least one non-terminal. Any language generated by a Type 0 Grammar is accepted by a Turing Machine.

Type I Grammar	A Type 0 Grammar (see above) with the restriction that the length of α must be less than or equal to the length of β , that is, $\beta \neq \lambda$. Any language generated by a Type I Grammar is accepted by an LBA.
VM	Java Virtual Machine.

APPENDIX B

TRADEMARK INFORMATION

Java	A registered trademark of Sun Microsystems, Inc.
Windows	A registered trademark of Microsoft Corporation

APPENDIX C

USER GUIDE FOR THE LBA TOOL

1. Introduction

The graphical LBA simulation tool is available at www.geocities.com/anonbeat to download. The tool will execute on any operating system platforms on which Java 2 SDK version 1.2 is installed.

The LBA tool provides a graphical environment to construct an LBA with a tape string and the animated view of the simulation of the LBA. The user interface of the tool was developed utilizing AWT and Swing components, which are parts of Java API. The main window of the LBA tool contains a variety of operations. Descriptions of all available actions are given in the next section.

2. File Menu

The File Menu contains options that allow a user to construct an LBA, store the LBA and the tape string, and retrieve a stored LBA and the corresponding tape string from files. A user can also print out a state diagram and terminate the tool through this menu.

The New menu item allows the user to construct a new LBA. If the current session is not empty, the current session will be replaced with the new session.

The Open File menu item allows the user to retrieve an existing LBA configuration file. When this item is selected, a file open dialog box pops up. This dialog box lists all the files with an lba extension in a directory. The directory can be changed using the Look option in the list box. This option provides a tree view of all directories and the user can double click on any of them to display the directory he/she needs. To open a file, one can double click on it or select it from the list and click on the Open button. If the selected file is in the correct format, the file is opened and the corresponding state diagram will be displayed in the state diagram window. At the same time, the input alphabet and the tape alphabet will be set. Then, the user can edit any of these configurations or simulate the

LBA. If the selected file is not in the required format or is corrupted, an error message box will pop up and inform the user as to the failure of the file retrieval process.

The Save As ... menu item allows the user to name and store the current LBA configurations in a new file as well as in an existing file. LBA configurations include the input alphabet, the tape alphabet, the end-markers, and the state diagrams. Any LBA configuration file requires a file extension of lba. If the user does not explicitly name a file with an lba extension, the tool automatically appends it to the file. If no desired directory exists, the user is able to create a new directory by clicking on the new directory button.

The Open Tape menu item performs the same routine as Open File except that it lists all files with a tpe extension that contain a tape string data. It also displays the tape window with the tape string that is read from a file.

The Save Tape As ... menu item performs the same routine as Save As ... except that it stores tape string data in the specified file.

The Print menu item allows the user to print out the LBA state diagram to the user-specified printer.

The Exit menu item terminates the current session of the LBA tool. When the user chooses this item, a popup window shows up to get confirmation from the user to complete the termination of the tool, and also to provide direct access to the online evaluation form. The user must respond to it by clicking one of the three buttons. If the user clicks on the YES button, the tool takes the user to the online evaluation form for the tool. At the completion or termination of the form, the tool will halt. If the user clicks on the NO button, the tool terminates immediately bypassing the evaluation phase. If the user chooses the last option, CANCEL, the tool cancels the termination.

3. Configure

The Configure menu provides four menu items: Set Input Alphabet, Set End Markers, Set Tape Alphabet, and Create/Edit Tape String. Each of these items must be selected for a new session. However, it is optional to the user whether or not he/she would like to open an LBA configuration file and a tape string file.

The Set Input Alphabet menu item allows the user to create a new input alphabet through the alphabet selection panel. The input alphabet is the domain of the read symbols. The user selects at least one symbol from the panel by clicking on the symbol that is to be selected. The OK button that is located on the bottom of the panel closes the panel and constructs the input alphabet. The input alphabet is used to create the transitions.

The Set End Markers menu item allows the user to choose the end-marker symbols. The user is required to select non-identical symbols for the left and right end-markers. The default end-markers are \$ as the left end-marker and ¢ as the right end-marker.

The Set Tape Alphabet menu item allows the user to create a new tape alphabet through the alphabet selection panel. The tape alphabet is the domain of the write symbols. The user selects any number of symbols from the panel by clicking on the symbols that are to be selected. The OK button that is located on the bottom of the panel closes the panel and constructs the tape alphabet. The tape alphabet is used to create the transitions.

The Create/Edit Tape String menu item provides the user with the ability to construct a new tape string using the tape string generator window (Figure 14). The user can specify the length of a particular symbol (e.g., if the current symbol is + and its length is 1, a + symbol will be appended to the right of the symbol 0 when the ADD button is clicked.) The DEL button can be clicked if the user needs to delete the current string. Deletion is based on the value of length. As many symbols as the value of length will be deleted from the right of the string. If the value of length is more than the actual length of the current string that is in the scrollable field, this field becomes empty. The Ok button is clicked to submit the current string, which is in the scrollable field, to be the tape string.

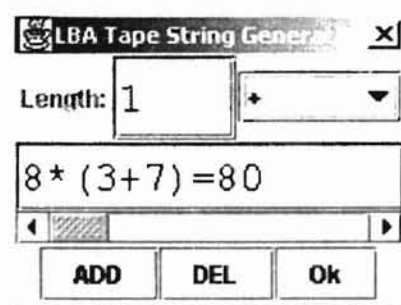


Figure 14. Tape string generator window

4. View

The View menu contains four checkbox menu items: Transition Table view option, Tree view option, Tape view option, and Speed Ctrl. Each menu item is associated with a checkbox. If a checkbox has a check mark, the user can uncheck it by clicking on it, and vice versa. The user can hide/show windows by clicking on these menu items.

5. Help

The Help menu provides online program documentation and Copy right information. Also it has a checkbox menu item that hides/shows the user guide window by checking or unchecking the box.

6. Icons

There are six operations that are available to the tool user. Clicking on the icons (STEP, UNDO, GO, PAUSE, ZOOM IN, and ZOOM OUT) initiates these operations. They are used for the simulation of the LBA. Except the ZOOM IN and ZOOM OUT icons, all the other icons are disabled in the edit mode.

The STEP operation performs one transition per click. One transition involves reading a tape symbol, changing the state, replacing the tape symbol, and moving the read/write head based on the direction parameter.

The UNDO operation undoes one transition per click. The undo buffer keeps all history of transitions the user makes; therefore, it is possible to undo back to the initial position.

The GO operation performs continuous STEP operations. To stop this operation, the PAUSE icon must be clicked.

The PAUSE operation breaks the current transition. The PAUSE icon is disabled until the GO operation is initiated. Additionally, while the GO operation is taking place, the GO icon is disabled.

The ZOOM IN and ZOOM OUT operations can be used to view the detail of the state diagram or the big picture, respectively. There are boundaries on the zoom operation. The operation is not affected when the ZOOM IN/OUT icon is clicked beyond the boundary.

7. RUN and EDIT Mode

There are two radio buttons in the main window that are labeled RUN and EDIT. Only one can be selected at a time. Initially, the EDIT button is selected that allows the user to construct an LBA or edit the current LBA. Once the RUN button is selected, the simulation icons will be enabled and the user can access these operations in order to simulate the LBA.

8. Edit Mode Operations

In the edit mode, the user is able to access the following operations: create a state, create a transition, move a state/transition, move selected area, delete a state, delete a transition, select the start state, select the final state, and edit a transition.

Under the create a state operation, clicking on the state diagram window draws a state with a state name. The state name is q followed by a number. The number is continuous from 0 to the maximum number of states - 1. The maximum number of states is set to 100. Thus, the user is not allowed to create more than 100 states.

The user is able to create a transition if there is at least one state in the window. In order to create a transition, first the user will click on the origination state, then the user will click on the destination state, and finally the user will click on the location on which the transition label is to appear.

The move a state/transition gives the user the ability to relocate states and transitions within the frame area by dragging the state and the transition label.

The move selected area option provides the user the ability to move multiple objects at one drag. To select a rectangular area, the user will point to the upper left corner and press the mouse button and then the user will drag to the lower right corner and release the button. Then, the rectangular area can be dragged.

The delete a state operation allows the user to remove the state by clicking on the state that is to be removed. Any transition from/to deleted state is automatically removed.

The delete a transition operation allows the user to remove a transition by clicking on the transition label.

The user can choose only one start state by clicking on the state when the select the start state option is selected. The start state is distinguished from other states by marking it with an arrowhead \triangleright .

More than one final state can be selected by clicking on the states. The user can click on a final state to reset it as a regular state.

The edit a transition option lets the user modify a transition label by clicking on the label that is to be modified. The user can edit a label through the edit transition window.

APPENDIX D

PROGRAMMER GUIDE FOR THE LBA TOOL

1. Description

The LBA simulation tool is a graphical tool that has been designed and implemented using an event-driven approach. The tool consists of Main Routine, User Interfaces, Event Listeners, and Event Handlers. The user-interface of the simulator was developed utilizing some of the Swing components. To run the tool, Java 2 SDK 1.2 is required. The UNIX environment also requires a CDE supported terminal.

The tool allows the user not only to simulate, but also to design, construct, and edit LBA configurations. The GUI environment provides the user easy access to the various features of the tool. Using the mouse to select the menu items from the menu bar or click on the icons in the main window triggers the tool's functions. The tool features permit the user to design an LBA configuration to be stored in a user specified file. This stored data can be retrieved and enhanced through editing its components (i.e., the alphabet and the state diagram) to modify the LBA.

2. Maintenance

The input alphabet and the tape alphabet will be stored in `inputAlphabet` and `tapeAlphabet` instances of `java.util.Hashtable`, respectively. These instance variables are located in the `LbaMainCtrlWindow.java` file. The domain of the alphabet and other constant values are stored in the `LbaData.java` file (i.e., domain of the end-markers, the menu names, the menu item names, the maximum number of states, the upper and lower bounds of speed, the URL for online documentation, the maximum number of transitions, and so on). These value can be changed carefully along with the system capacity.

The state diagram information is divided into five different parts to be stored. These parts are: state representation, index of states, transition label area, transition label, and transition location objects. They are stored in `stateV`, `indexV`, `labelV`, `transV`, and `fromToV` `java.util.Vector` instances.

The state representation object holds the size of the state and the coordinates of the state location. The index of the states object keeps index data for the states such as q_0 and q_1 . The number of these objects in $stateV$ and $indexV$ must be the same at any time.

The transition label area is represented by an invisible rectangular object which holds its size and location. A transition label is a string object that is displayed on the transition. The transition location object contains two state indexes: one holds the originated state index and the other holds the destination state index. This information is used for drawing an arrow between the states. The number of these objects in $labelV$, $transV$, and $fromToV$ must be the same at any time.

APPENDIX E

EVALUATION FORM

1. Platform – Choose one of the following that you used to run the simulator:
 - a. Windows 95
 - b. Windows 98
 - c. Windows NT
 - d. Windows 2000 Professional
 - e. Windows 2000 Server/Advanced server
 - f. UNIX from a terminal running the CDE environment
 - g. UNIX from the X-server environment under Windows 95
 - h. UNIX from the X-server environment under Windows 98
 - i. UNIX from the X-server environment under Windows NT
 - j. UNIX from the X-server environment under Windows 2000 Pro.
 - k. UNIX from the X-server environment under Windows 2000 Server
 - i. None of the above
2. Accessibility – Choose one of the following:
 - a. It is difficult to use the tool.
 - b. I have no idea what to do with the tool.
 - c. The tool is easy to use.
3. Speed of Processing
 - a. Initialization speed – Choose one of the following:
 - i. Very slow
 - ii. Slow
 - iii. Ok
 - iv. Fast
 - v. Very fast
 - b. Animation of mapping speed – Choose one of the following:
 - i. Very slow
 - ii. Slow
 - iii. Ok
 - iv. Fast
 - v. Very fast
 - c. Editing mode speed – Choose one of the following:

- i. Very slow
- ii. Slow
- iii. Ok
- iv. Fast
- v. Very fast

4. Necessity – Rank the necessity of the following items on a scale of 1 through 5 (1 is not necessary and 5 is quite necessary).

a.	Save/open file	1	2	3	4	5
b.	Zoom in/out featur	1	2	3	4	5
c.	Speed controler	1	2	3	4	5
d.	Process tree view	1	2	3	4	5
e.	Transition function table	1	2	3	4	5
f.	Go function	1	2	3	4	5
g.	Step function	1	2	3	4	5
h.	Stop function	1	2	3	4	5

5. Overall the simulator is

- a. Very useful
- b. Useful
- c. Not useful
- d. Useless

6. The tool was designed as a pedagogical aid (i.e., for educational purposes). What is your opinion about the tool?

- a. It is absolutely helpful.
- b. It is helpful.
- c. It is not so helpful.
- d. It helps is confusing.

7. Bug report – While you were running the tool, did you face any of the following errors?

- a. The computer was froze
- b. The computer gave you a system error message
- c. The computer became very slow
- d. The tool returned a programming error
- e. You encountered other types of errors

8. Future plan – Would you like to use the simulator in the future?

- a. Yes, I do.
- b. If the tool can be overcome the above problems, I would like to use the improved version.
- c. No, I don't.

APPENDIX F

SELECTED PROGRAM LISTINGS

The implementation of the LBA tool is discussed in Section 3.2. What follows is the list of all source files in the implementation of the tool.

QuadraticForm.java	LbaCtrlSpeed.java
MultiSpanCellTableUI.java	LbaCtrlProcess.java
MultiSpanCellTable.java	LbaCtrlLabel.java
Location.java	LbaCtrlGraphics.java
LbaTreeView.java	LbaClone.java
LbaTiledWindow.java	LbaAlphabetPanel.java
LbaThread.java	JFtp.java
LbaTapePanel.java	GroupableTableHeaderUI.java
LbaSelectionPanel.java	GroupableTableHeader.java
LbaPopup.java	Format.java
LbaMainCtrlWindow.java	DefaultCellAttribute.java
LbaMain.java	Curves.java
LbaHelp.java	ColumnGroup.java
LbaGuide.java	ColoredCell.java
LbaFileOut.java	CellSpan.java
LbaFileIn.java	CellFont.java
LbaEndmarkerPanel.java	CellDefaultAttribute.java
LbaEditTransition.java	CellColor.java
LbaDrawArea.java	CellAttributiveTableModel.java
LbaData.java	CellAttributiveRenderer.java
LbaCtrlZoom.java	CellAttribute.java
LbaCtrlTable.java	AttributiveCellTableModel.java

The total number of classes is 72. The total number of source files is 44. The total number of lines in these source files is approximately 10,000. The list of the selected program listings in this appendix follows:

LbaDrawArea.java
LbaCtrlProcess.java
LbaData.java

LbaCtrlZoom.java
 CellAttributiveRenderer.java (Author: Nobuo Tamemasa)
 QuadraticForm.java

```

/**
 * @author Joji Doi
 * @date 11/13/2000
 * @file LbaDrawArea.java
 * @class LbaDrawArea
 * @innner class BasicActionListener
 */
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Rectangle;

import java.awt.geom.Arc2D;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.awt.geom.QuadCurve2D;
import java.awt.geom.Rectangle2D;

import java.awt.print.PageFormat;
import java.awt.print.Printable;

import java.util.Stack;
import java.util.Vector;
import java.util.Hashtable;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

/**
 * Contains the drawing area for the state diagram.
 * The inner class is used for event listener perpose.
 * Vector instance variables manages state diagram data.
 */
public class LbaDrawArea extends JPanel implements Printable{

    private boolean NoError; // error checking at Assertions.

    private LbaEditTransition LET; //new interface

    /**
     * Ellipse object. for the state representation.
     * indexV.size() == stateV.size()
     */
    private Vector stateV;

    /**
     * Integer object. for state index i.e. q0, q1
     * indexV.size() == stateV.size()
     */
    private Vector indexV;

```

```

/**
 * Rectangle2D object. for label area on the transition
 * labelV.size() == transV.size() == fromToV.size()
 */
private Vector labelV;

/**
 * String of "x/y D" and length is always 5.
 * labelV.size() == transV.size() == fromToV.size()
 */
private Vector transV;

/**
 * int[2] has state index a[0] == from state index
 * a[1] == to state index and they are found in indexV.
 * labelV.size() == transV.size() == fromToV.size()
 */
private Vector fromToV;

private int size;

//assigned at editTransition used at actionPerformed
private int editIndex;

/**
 * Integer for state index that is found in indexV.
 */
private Vector finalV;

public Integer startState;

private boolean [] bOccupied;

// used for paint routine.
//
private int iFrontState;
private int iToState;
private Point mouseP;

// set it in LbaEditTransition
// ActionListener is in this class.
private JButton inputB;
private JButton tapeB;
private JButton directionB;
private JButton b3in1; // do it all.
private JButton bEdit; // editing transition ok button

//temporary buffer for a String of (x/y D)
//and it will be an element of transV.
private String sTransiion;

//public boolean [] InputAlphabet, TapeAlphabet;

public Hashtable inputAlphabet, tapeAlphabet;
public Character [] Endmarkers;

public Integer currentState;

private boolean bSelect = true;
public boolean bVisibleSelectingArea;
public Point dragAreaStartPoint; // buffer for start p

```

```

public Point dragAreaEndPoint; // buffer for end p

/**
 * Constructor for this class initialize instance variables.
 * And set buttons to action listener.
 */
public LbaDrawArea(){

    bOccupied = new boolean[LbaData.MaxStates];

    NoError = true;

    stateV = new Vector();
    indexV = new Vector();
    labelV = new Vector();
    transV = new Vector();
    fromToV = new Vector();
    size = 2;

    finalV = new Vector();
    startState = new Integer(-1);
    currentState = new Integer(-1);
    iFrontState = -1;
    iToState = -1;
    mouseP = null;

    //Create Option buttons
    //
    inputB = new JButton("Ok");
    tapeB = new JButton("Ok");
    directionB = new JButton("Ok");

    bEdit = new JButton("Ok");
    b3in1 = new JButton("Ok");

    // ActionListener components.
    // buttons, radiobuttons, menuItems, comboBox
    BasicActionListener bActionL =
        new BasicActionListener();
    inputB.addActionListener(bActionL);
    tapeB.addActionListener(bActionL);
    directionB.addActionListener(bActionL);
    b3in1.addActionListener(bActionL);
    bEdit.addActionListener(bActionL);

} //const.

/**
 * This constructor is used to copy the other instance
 * of this class by its contents. It is not copy of
 * reference. This constructor is used for
 * the non-deterministic LBA.
 */
public LbaDrawArea( LbaDrawArea LDA){

    bOccupied = (boolean []) ((LDA.bOccupied).clone());

    NoError = true;

    stateV = LbaClone.vector((Vector)LDA.stateV);
    indexV = (Vector)LDA.indexV.clone();

```

```

labelV = LbaClone.vector((Vector)LDA.labelV);
transV = (Vector)LDA.transV.clone();
fromToV = LbaClone.vector((Vector)LDA.fromToV);

size = LDA.size;

//Shared vector
//
finalV = LDA.finalV;
startState = LDA.startState;
currentState = new Integer(LDA.currentState.intValue());

iFrontState = -1;
iToState = -1;
mouseP = null;

//Create Option buttons
//
inputB = new JButton("Ok");
tapeB = new JButton("Ok");
directionB = new JButton("Ok");

bEdit = new JButton("Ok");
b3in1 = new JButton("Ok");

//Hashtable for the selected alphabet
//These can be shared among frames.
//
inputAlphabet = LDA.inputAlphabet;
tapeAlphabet = LDA.tapeAlphabet;
Endmarkers = LDA.Endmarkers;

// ActionListener components.
// buttons, radiobuttons, menuItems, comboBox
BasicActionListener bActionL =
    new BasicActionListener();
inputB.addActionListener(bActionL);
tapeB.addActionListener(bActionL);
directionB.addActionListener(bActionL);
b3in1.addActionListener(bActionL);
bEdit.addActionListener(bActionL);

} //const.

/**
 * Abstract method for the interface Printable definition.
 * This method is used to print the state diagram to the
 * user specifying printer.
 */
public int print(Graphics g, PageFormat pageformat, int i){

    int result = Printable.NO_SUCH_PAGE;

    if(i < 1){

        result = Printable.PAGE_EXISTS;
        Graphics2D g2 = (Graphics2D)g;
        g2.setBackground(Color.white);
        g2.translate(pageformat.getImageableX(),
            pageformat.getImageableY());
    }
}

```

```

        setGraphics2D(g2);

    }//if

    return result;

} //print

/**
 * Overriding method. Actual processing is done in
 * setGraphics2D method. The reason of this method call
 * is print method also invokes setGraphics2D method.
 */
public void paint(Graphics g){

    Graphics2D g2 = (Graphics2D)g;
    //setBackground(Color.white); -> JScrollPane needs to be set.
    setGraphics2D(g2);

} //paint

/**
 * Actual implementaion of drawing object on the canvas.
 * This is invoked from paint and print methods. It draws
 * the states, the transitions, the labels, the start state,
 * and the final states.
 */
private void setGraphics2D(Graphics2D g2){

    //Set the size of font
    //
    g2.setFont(
        new Font("Courier", Font.PLAIN, LbaData.aFontSize[size]));

    //Draw states and states' name on states.
    //
    for(int i=0;i<stateV.size();i++){

        Integer iTemp = (Integer)indexV.elementAt(i);

        //Current state symbol
        //
        if(0 == iTemp.compareTo(currentState)){
            g2.setColor(Color.blue);
        } //if

        Ellipse2D.Double e =
            (Ellipse2D.Double)stateV.elementAt(i);
        g2.draw(e);

        g2.drawString(LbaData.StateIndex + iTemp,
            (float)(e.getX() + e.getWidth()/(i<10?4:8)),
            (float)(e.getY() + (5*e.getWidth())/8));

        if(0 == iTemp.compareTo(currentState)){
            g2.setColor(Color.black);
        } //if

        //Start state symbol
        //

```

```

if(0== iTemp.compareTo(startState)){

    Point2D.Double point1 =
        new Point2D.Double(
            e.getX(), e.getY() + e.getHeight()/2);

    Point2D.Double point2 =
        new Point2D.Double(
            point1.getX() - e.getWidth()/3,
            point1.getY() - e.getHeight()/3);

    Point2D.Double point3 =
        new Point2D.Double(
            point1.getX() - e.getWidth()/3,
            point1.getY() + e.getHeight()/3);

    g2.draw(new Line2D.Double(point1, point2));
    g2.draw(new Line2D.Double(point1, point3));
} //if
} //for

/**
 * Draw final states
 */
for(int i=0;i<finalV.size();i++){
    int index = ((Integer)finalV.elementAt(i)).intValue();

    for(int j=0;j<indexV.size(); j++){
        if( index == ((Integer)indexV.elementAt(j)).intValue()){

            Ellipse2D.Double e =
                (Ellipse2D.Double)stateV.elementAt(j);

            g2.draw(new Ellipse2D.Double(
                e.getX() + e.getWidth() * .1,
                e.getY() + e.getHeight() * .1,
                e.getWidth() - e.getWidth() * .2,
                e.getHeight() - e.getHeight() * .2));

        } //if
    } //for
} //for

//Assertion
NoError = ( transV.size() == labelV.size() &&
    fromToV.size() == transV.size() )
    ?true:false;

//Draw labels in Rectangle2D and draw curves
//between the two states.
//
for(int i=0;NoError && i<transV.size();i++){

    //Draw a rect. for label
    //
    Rectangle2D.Double r =
        (Rectangle2D.Double)labelV.elementAt(i);
    //g2.draw(r);

    int [] stateIndex = (int[])fromToV.elementAt(i);

    //These variables will have their values depending on
    //arrowhead state condition.
    //
    Point2D.Double labelLoc;

```



```

Point ToP;
Point E;

if(stateIndex[0] != stateIndex[1]){

    //From State != To State
    //
    double radius = (double)(getStateSize(stateIndex[0]))/2;
    labelLoc =
        new Point2D.Double(r.getX(), r.getY());
    E = new Point((int)r.getX(), (int)r.getY());

    //Get two points of the states.
    //
    Point FromP = new Point(
        new QuadraticForm(
            Location.getMiddleP(getStatePoint(stateIndex[0])),
            getStateSize(stateIndex[0])),
        labelLoc, radius).getIntersection());

    ToP = new Point(
        new QuadraticForm(
            Location.getMiddleP(getStatePoint(stateIndex[1])),
            getStateSize(stateIndex[1])),
        labelLoc, radius).getIntersection());

    /**
     * The following algorithm enables labels to be
     * on transition curbs.
     */
    Point A = FromP;
    Point B = ToP;
    Point C = new Point((int)labelLoc.x, (int)labelLoc.y);

    double distAB = Location.getLength(A,B);

    QuadraticForm qf = new QuadraticForm(A,B,distAB/2);
    Point D = qf.getIntersection();

    double distCD = Location.getLength(C,D);
    qf = new QuadraticForm(C,D,distCD,true);
    E = qf.getIntersection();

    //Draw a curve between the two states.
    //
    QuadCurve2D.Double q =
        new QuadCurve2D.Double( FromP.x, FromP.y,
            E.x, E.y, ToP.x, ToP.y);
    g2.draw(q);

} //if
else{

    //From State == To State
    //
    Point p = getStatePoint(stateIndex[0]);
    int d = getStateSize(stateIndex[0]);

    //Draw an arc on the state.
    //
    Arc2D.Double a =
        new Arc2D.Double(
            p.getX()-d*.2,
            p.getY()-d,
            d+d*.4,
            d+d*.4,

```

```

        315,
        270,
        Arc2D.OPEN);
g2.draw(a);

//Arrowhead computation
//
double centerX = a.getX() + a.getWidth()/2;
double centerY = a.getY() + a.getHeight()/2;

labelLoc =
    new Point2D.Double(a.getX()-d*.3,a.getY());

E = new Point((int)(a.getX()-d*.3), (int)a.getY());

ToP = new Point(
    (int)(centerX - (a.getWidth()/(2*Math.sqrt(2)))),
    (int)(centerY + (a.getWidth()/(2*Math.sqrt(2)))));
}

//Draw an arrowhead
//
////////////////////////////////////
//
// Prepare drawing arrow head.
//
////////////////////////////////////
//Arrowhead size is the same as font size
//
double length = LbaData.aFontSize[size];

/**
 * The reference for the following mathematical
 * formula to draw an arrowhead is:
 * Muhammad T. Hassan, "Towards a Graphical Petri Net Tool",
 * Master of Science Thesis, Computer Science Department,
 * Oklahoma State University, Stillwater, OK, 1993.
 */
//XXX Center of each state. ( pFrom.x, pFrom.y )
//XXX ( pTo.x, pTo.y )
double alpha = Math.atan2(
    E.y - ToP.getY(),
    E.x - ToP.getX());
//constant
final double beta = 20.0 * 0.01745329;
Point2D.Double p1 =
    new Point2D.Double(
        ToP.x + (length * Math.cos(alpha+beta)),
        ToP.y + (length * Math.sin(alpha+beta)));

Point2D.Double p2 =
    new Point2D.Double(
        ToP.x + (length * Math.cos(alpha-beta)),
        ToP.y + (length * Math.sin(alpha-beta)));

//XXX Add scroll value.
//XXX Draw arrow heads.
//XXX An arrow head consists of two lines.

g2.draw( new Line2D.Double(
    new Point2D.Double(ToP.x, ToP.y), p1));
g2.draw( new Line2D.Double(
    new Point2D.Double(ToP.x, ToP.y), p2));

//Draw a label

```

```

        g2.drawString((String)transV.elementAt(i),
            (float)(r.getX() + r.getWidth()/8),
            (float)(r.getY() + (5*r.getWidth())/18));
    }

    //draw a selecting area
    if(bVisibleSelectingArea &&
        dragAreaStartPoint != null &&
        dragAreaEndPoint != null){

        int X = dragAreaStartPoint.x < dragAreaEndPoint.x
            ?dragAreaStartPoint.x:dragAreaEndPoint.x;
        int Y = dragAreaStartPoint.y < dragAreaEndPoint.y
            ?dragAreaStartPoint.y:dragAreaEndPoint.y;
        int W =
            Math.abs(dragAreaStartPoint.x - dragAreaEndPoint.x);
        int H =
            Math.abs(dragAreaStartPoint.y - dragAreaEndPoint.y);

        g2.draw( new Rectangle(X,Y,W,H));

    }

    }

    //setGraphics2D [for paint and print routine]

    /**
     * Moves any states and transitions with in the
     * selected area
     */
    public void moveObjects(int xMove, int yMove){

        int Xsmaller, Xlarger, Ysmaller, Ylarger;
        if(dragAreaStartPoint.x < dragAreaEndPoint.x){
            Xsmaller = dragAreaStartPoint.x;
            Xlarger = dragAreaEndPoint.x;
        }
        else{
            Xsmaller = dragAreaEndPoint.x;
            Xlarger = dragAreaStartPoint.x;
        }

        if(dragAreaStartPoint.y < dragAreaEndPoint.y){
            Ysmaller = dragAreaStartPoint.y;
            Ylarger = dragAreaEndPoint.y;
        }
        else{
            Ysmaller = dragAreaEndPoint.y;
            Ylarger = dragAreaStartPoint.y;
        }

        //find for states
        for(int i=0;i<stateV.size();i++){

            Ellipse2D.Double e =
                (Ellipse2D.Double)stateV.elementAt(i);

            if( Xsmaller < e.getX() && e.getX() < Xlarger &&
                Ysmaller < e.getY() && e.getY() < Ylarger){

                e setFrame(
                    xMove + e.getX(), yMove + e.getY(),
                    e.getWidth(), e.getHeight());
            }
        }
    }

```

```

    }//if

    }//for
    for(int i=0;i<labelV.size();i++){
        Rectangle2D.Double r =
            (Rectangle2D.Double)labelV.elementAt(i);

        if( Xsmaller < r.getX() && r.getX() < Xlarger &&
            Ysmaller < r.getY() && r.getY() < Ylarger){

            r.setRect(
                xMove + r.getX(), yMove + r.getY(),
                r.getWidth(), r.getHeight());

        }//if

    }//for

    dragAreaStartPoint = new Point(
        dragAreaStartPoint.x + xMove,
        dragAreaStartPoint.y + yMove);

    dragAreaEndPoint = new Point(
        dragAreaEndPoint.x + xMove,
        dragAreaEndPoint.y + yMove);

    }//moveObjects

    /**
     * Returns transV and fromToV references.
     * The return array object is length of 2
     */
    public Vector [] getRange(){
        Vector [] v = new Vector[2];
        v[0] = transV;
        v[1] = fromToV;

        return v;
    }//getRange

    /**
     * Returns all of the state indexes.
     * The return array of String is length of indexV's size.
     */
    public String [] getStates(){
        String [] states = new String[indexV.size()];

        for(int i=0;i<states.length;i++){

            states[i] = LbaData.StateIndex +(Integer)indexV.elementAt(i);
        }//for
        return states;
    }//getStates

    /**
     * The accepting object is the instance of LbaFileOut.
     * This method stores all LBA data to the file through
     * the LbaFileOut instance.
     */
    public void saveData(Object o){

```

```

LbaFileOut out = (LbaFileOut)o;

Vector AstateV = new Vector();
for(int i=0;i<stateV.size();i++){
    double [] dTemp = new double[4];
    dTemp[0] = ((Ellipse2D.Double)stateV.elementAt(i)).getX();
    dTemp[1] = ((Ellipse2D.Double)stateV.elementAt(i)).getY();
    dTemp[2] =
        ((Ellipse2D.Double)stateV.elementAt(i)).getWidth();
    dTemp[3] =
        ((Ellipse2D.Double)stateV.elementAt(i)).getHeight();

    AstateV.addElement(dTemp);
} //for
out.write(AstateV);
out.write(indexV);

Vector AlabelV = new Vector();
for(int i=0;i<labelV.size();i++){
    double [] dTemp = new double[4];
    dTemp[0] = ((Rectangle2D.Double)labelV.elementAt(i)).getX();
    dTemp[1] = ((Rectangle2D.Double)labelV.elementAt(i)).getY();
    dTemp[2] =
        ((Rectangle2D.Double)labelV.elementAt(i)).getWidth();
    dTemp[3] =
        ((Rectangle2D.Double)labelV.elementAt(i)).getHeight();

    AlabelV.addElement(dTemp);
} //for
out.write(AlabelV);
out.write(transV);
out.write(fromToV);

//Save size value and start state and tape alphabet
//input alphabet.
//
Vector vOthers = new Vector();
vOthers.addElement(new Integer(size));
vOthers.addElement(startState);
vOthers.addElement(inputAlphabet);
vOthers.addElement(tapeAlphabet);
vOthers.addElement(Endmarkers);
vOthers.addElement(bOccupied);
out.write(vOthers);

//Save final states
//
out.write(finalV);

} //saveData

/**
 * The accepting object is the instance of LbaFileIn.
 * This method retrieves all LBA data from the file through
 * the LbaFileIn instance.
 */
public boolean loadData(Object o){

    LbaFileIn in = (LbaFileIn)o;

    Vector AstateV = in.read();

    stateV.removeAllElements();
    for(int i=0;i<AstateV.size();i++){

```

```

        double [] dTemp = (double []) AstateV.elementAt(i);

        Ellipse2D.Double eTemp = new Ellipse2D.Double(
            dTemp[0], dTemp[1], dTemp[2], dTemp[3] );

        stateV.addElement(eTemp);
    } //for

    indexV = in.read();

    Vector AlabelV = in.read();
    labelV.removeAllElements();
    for(int i=0; i<AlabelV.size(); i++){

        double [] dTemp = (double []) AlabelV.elementAt(i);

        Rectangle2D.Double rTemp = new Rectangle2D.Double(
            dTemp[0], dTemp[1], dTemp[2], dTemp[3] );
        labelV.addElement(rTemp);
    } //for

    transV = in.read();
    fromToV = in.read();

    Vector vOthers = in.read();
    if(vOthers.size()==6){
        size = ((Integer)vOthers.elementAt(0)).intValue();
        startState = (Integer)vOthers.elementAt(1);
        if(vOthers.elementAt(2) instanceof Hashtable){
            inputAlphabet = (Hashtable)vOthers.elementAt(2);
            tapeAlphabet = (Hashtable)vOthers.elementAt(3);

            } //if
        else{
            //old version.
            System.out.println("Old version.");
            inputAlphabet = new Hashtable();
            tapeAlphabet = new Hashtable();
        } //else
        Endmarkers = (Character [])vOthers.elementAt(4);
        bOccupied = (boolean [])vOthers.elementAt(5);
    } //if
    finalV = in.read();

    currentState = startState;

    return in.bError;
} //loadData

/**
 * Draws start state and final state which process endmarkers.
 * This is invoked only for new LBA construction.
 */
public void setDefault(){

    //start state
    Point pt = new Point(100,200);
    setState(pt);
    bSelect = false;
    setTransition(pt,pt,new Point(pt.x-20,pt.y-40));

    sTransion = ""+Endmarkers[0]+"/"+Endmarkers[0]+" R";
    setTransitionLabel();
    bSelect = true;
    setStart(pt);

```

```

//end states and transition
Point ptFr = new Point (300, 200);
Point ptTo = new Point (400, 200);
setState(ptFr);
setState(ptTo);
bSelect = false;
setTransition(ptFr,ptTo,new Point ((ptFr.x+ptTo.x)/2-20,ptFr.y));

sTransiion = ""+Endmarkers[1]+"/"+Endmarkers[1]+" S";
setTransitionLabel();
bSelect = true;
setFinal(ptTo);

} //setDefault

/**
 * Initialize state diagram information storage to be
 * empty.
 * The current data will be lost after this operation.
 */
public void setNew(){

    bOccupied = new boolean[LbaData.MaxStates];

    NoError = true;

    stateV = new Vector();
    indexV = new Vector();
    labelV = new Vector();
    transV = new Vector();
    fromToV = new Vector();
    size = 2;

    finalV = new Vector();
    startState = new Integer(-1);
    currentState = new Integer(-1);
    iFrontState = -1;
    iToState = -1;
    mouseP = null;

} //setNew

/**
 * Store transition label to labelV from accepting object.
 */
public void setText( String s ){
    labelV.addElement(s);
} //setText

/**
 * If accepting coordinate matches with any state,
 * mark it as the start state.
 */
public void setStart(Point p){

    //index for stateV not for state itself.
    //
    int index4sV = Location.hasState(stateV, p);

    //Assertion
    NoError = stateV.size() == indexV.size()
        ?true:false;

    if(index4sV!=-1 && NoError){
        startState = new Integer( index4sV );
    }

```

```

        currentState = startState;

    }//if

} //setStart

/**
 * If the accepting coordinate matches with any state,
 * mark it as the final state. If it is already the
 * final state, mark it as non final state.
 */
public void setFinal(Point p){

    //index for stateV not for state itself.
    //
    int index4sV = Location.hasState(stateV, p);

    //Assertion
    NoError = stateV.size() == indexV.size()
        ?true:false;

    if(index4sV!=-1 && NoError){
        boolean quit = false;

        for(int i=0;!quit && i<finalV.size();i++){

            if( index4sV ==
                ((Integer)finalV.elementAt(i)).intValue()){

                quit = true;
                finalV.removeElementAt(i);
            }//if

        }//for
        if(!quit)
            finalV.addElement( new Integer( index4sV ));
    }//if
} //setFinal

/**
 * This method adds states to the stateV if the number
 * of states does not exceed maximum number of states.
 * The maximum number of states is MaxStates constant.
 */
public void setState(Point p){

    if( stateV.size() >= LbaData.MaxStates ){

        LbaPopup.Error(8);

    }//if overflow
    else{
        stateV.addElement(
            new Ellipse2D.Double(
                p.x - LbaData.aStateSize[size]/2,
                p.y - LbaData.aStateSize[size]/2,
                LbaData.aStateSize[size],
                LbaData.aStateSize[size]));

        indexV.addElement( new Integer( findIndex() ));

    }//else

```



```

} //setState

/**
 * Returns true if the accepting coordinate matches
 * with any state in the canvas. Otherwise returns false.
 */
public boolean accept(Point p){

    return Location.hasState(stateV, p) != -1;

} //accept

/**
 * Create new instance of transition with these
 * accepting objects.
 */
public void setTransition(Point FromP, Point ToP,
    Point LabelP){

    //check FromP and ToP match to any states.
    //
    int indexFP = Location.hasState(stateV, FromP);
    int indexTP = Location.hasState(stateV, ToP);

    //keep the mouse pointer location and state index
    iFrontState = indexFP;
    iToState = indexTP;
    mouseP = new Point(LabelP);

    //Selection dialog window.
    //
    if(bSelect)
        setVisible(b3in1);

} //setTransition

/**
 * The first parameter decides if it is zoom in
 * or zoom out operation. The second parameter is
 * the coordinate of the center position.
 * The coordinate can be user specified for future
 * version of the tool.
 */
public void setZoom(boolean IsZoomIn, Point p){
    Point2D.Double frameCenterP =
        new Point2D.Double(p.x, p.y);

    //aStateSize.length and aStringSize.length and
    //C_zoomStateSize.length must be equal.
    //
    if(IsZoomIn){
        if(size < LbaData.aStateSize.length-1){
            ++size;

            LbaCtrlZoom.IN(stateV, labelV, frameCenterP, size);
        } //if
    } //if
    else{
        if(size > 0){
            --size;
            LbaCtrlZoom.OUT(stateV, labelV, frameCenterP, size);
        } //if
    } //else
}

```

```

} //setZoom

/**
 * This method deletes the specified state as well as
 * related labels.
 */
public void delState(Point p){

    //index for stateV not for state itself.
    //
    int index4sV = Location.hasState(stateV, p);

    //Assertion
    NoError = stateV.size() == indexV.size()
        ?true:false;

    if( NoError && index4sV >= 0 && index4sV < stateV.size()){

        //Assertion
        NoError = ( transV.size() == labelV.size() &&
            fromToV.size() == transV.size() )
            ?true:false;

        //remove corresponded transition
        //exhausted search for multiple deletion.
        //
        for( int i=0; NoError && i<transV.size();){

            int [] index4s = (int[])fromToV.elementAt(i);

            //findIndex for stateV by state label
            if(findIndex(index4s[0]) == index4sV
                || findIndex(index4s[1]) == index4sV){

                // i does not need to be incremented.
                transV.removeElementAt(i);
                labelV.removeElementAt(i);
                fromToV.removeElementAt(i);

            } //if
            else{
                ++i; // incrementation is needed in here not in if.
            } //else
        } //for

        /**
         * change occupancy flag to be available and
         * remove index data and final state data.
         */
        stateV.removeElementAt(index4sV);

        bOccupied[(((Integer)indexV.
            elementAt(index4sV)).intValue())] = false;

        boolean quit = false;
        for(int i=0; i<finalV.size() && !quit; i++){
            if(0==(((Integer)indexV.elementAt(index4sV)).compareTo(
                (Integer)(finalV.elementAt(i))))){

                finalV.removeElementAt(i);
                quit = true;
            }
        }
    }
}

```

```

        }//if
    }//for

    indexV.removeElementAt(index4sV);

    }//if
    else{
        // cannot delete any state. click right on.
    }//else
}//delState

/**
 * Delete a transition if Point p points any of
 * existing label area, Rectangle2D.Double.
 */
public void delTransition(Point p){

    //Assertion
    NoError = ( transV.size() == labelV.size() &&
        fromToV.size() == transV.size() )
        ?true:false;

    //This loop deletes all transitions if they are overlapped.
    //To make this deletes just one transition per click
    //add boolean quit variable, or make this choice depending
    //on the user, then add checkBox instance in somewhere.
    //
    for( int i=0; NoError && i<transV.size(); ){

        Rectangle2D.Double r =
            (Rectangle2D.Double)labelV.elementAt(i);

        //true if mouse point p is within the Rectangle instance
        //
        if(p.getX() >= r.getX() &&
            p.getX() <= r.getX() + r.getWidth() &&
            p.getY() >= r.getY() &&
            p.getY() <= r.getY() + r.getHeight()){

            // i does not need to be incremented.
            transV.removeElementAt(i);
            labelV.removeElementAt(i);
            fromToV.removeElementAt(i);

        }//if
        else{
            ++i; // incrementation is needed in here not in if.
        }//else
    }//for
}//delTransition

/**
 * Change state location data based on mousePoint value.
 * The first parameter is the index for stateV. The
 * second parameter is the new location of the state.
 */
public void moveState(int index4sV, Point mousePoint){

```

```

//Assertion
NoError = stateV.size() == indexV.size()?true:false;

if(!NoError){
    System.out.println("Error @ LbaDrawArea.moveState");
    System.exit(102);
}

//error

Ellipse2D.Double e =
    (Ellipse2D.Double)stateV.elementAt(index4sV);

int newX = mousePoint.x < 0?0:mousePoint.x;
int newY = mousePoint.y < 0?0:mousePoint.y;

Point newP = Location.getCenterP(
    new Point(newX,newY), (int)e.getWidth());

e setFrame(
    newP.getX(), newP.getY(), e.getWidth(), e.getHeight());
}

//moveState

/**
 * Change transition location data based on mousePoint value.
 * The first parameter is the index for labelV. The
 * second parameter is the new location of the label.
 */
public void moveTrans(int index4tV, Point mousePoint){

    //Assertion
    NoError = ( transV.size() == labelV.size() &&
        fromToV.size() == transV.size() )
        ?true:false;

    if(!NoError){
        System.out.println("Error @ LbaDrawArea.moveTrans");
        System.exit(103);
    }

    //error

    Rectangle2D.Double r =
        (Rectangle2D.Double)labelV.elementAt(index4tV);

    int newX = mousePoint.x < 0?0:mousePoint.x;
    int newY = mousePoint.y < 0?0:mousePoint.y;

    //creates newP top-left-corner of the rectangle
    //based on the mouse clicked location.
    //
    Point2D.Double newP = new Point2D.Double(
        newX - r.getWidth()/2,
        newY - r.getHeight()/2);

    r.setRect(newP.getX(), newP.getY(),
        r.getWidth(), r.getHeight());
}

//moveTrans

/**

```

```

* Returns true if curState is one of the final states.
* Otherwise, returns false.
*/
public boolean checkFinalState(int curState){

    boolean quit=false;
    for(int i=0;!quit && i<finalV.size();i++){
        quit =
            curState == ((Integer)finalV.elementAt(i)).intValue();
    }//for

    return quit;

}

/**
 * If the accepting coordinate matches with any of
 * the transition label, popups a edit window for
 * editing the transition label.
 */
public void editTransition(Point p){

    boolean quit = false;
    for( int i=0; !quit && NoError && i<transV.size();){

        Rectangle2D.Double r =
            (Rectangle2D.Double)labelV.elementAt(i);

        //true if mouse point p is within the Rectangle instance
        //
        if(p.getX() >= r.getX() &&
            p.getX() <= r.getX() + r.getWidth() &&
            p.getY() >= r.getY() &&
            p.getY() <= r.getY() + r.getHeight()){

            // i does not need to be incremented.
            String label = (String)transV.elementAt(i);

            editIndex = i;
            LET = new LbaEditTransition(bEdit,inputAlphabet,
                tapeAlphabet, Endmarkers, label);
            LET.setVisible(true);

            quit = true;

        }//if
        else{
            ++i; // incrementation is needed in here not in if.
        }//else
    }//for

}

/**
 * Find out which index is not occupied. If it is found
 * it will be returned.
 */
private int findIndex(){
    boolean quit = false;
    int index = -1;

    for(int i=0;!quit && i<bOccupied.length;i++){
        index = !bOccupied[i]?i:index;
        quit = index==i?true:false;
    }
}

```

```

    }//for

    if(index!=-1)
        bOccupied[index] = true;

    return index;

} //findIndex

/**
 * Find an index4sV( index of Vector indexV) if the index found
 */
private int findIndex(int stateIndex){
    int vectorIndex = -1;
    boolean quit = false;

    //Assertion
    //
    NoError = stateV.size() == indexV.size()
        ?true:false;

    //Search by index.
    //
    for(int i=0;!quit && NoError && i<indexV.size(); i++){
        int iTemp = ((Integer)indexV.elementAt(i)).intValue();

        if(stateIndex == iTemp){
            vectorIndex = i;
            quit = true;
        }
    } //if
} //for

if(vectorIndex == -1){
    System.out.println("Error @ LbaDrawArea.findIndex");
    System.exit(101);
}

return vectorIndex;

} //findIndex

/**
 * Returns the index of stateV, if the accepting
 * coordinate matches with any of existing states.
 * Otherwise, returns -1.
 */
public int findIndex( Point p ){

    return Location.hasState(stateV, p);

} //findIndex

/**
 * Returns the index of labelV, if the accepting
 * coordinate matches with any of existing label.
 * Otherwise, returns -1.
 */
public int findIndex4tV(Point p){

    boolean quit = false;
    int index4tV = -1;

    //Assertion
    NoError = ( transV.size() == labelV.size() &&

```

```

        fromToV.size() == transV.size() )
        ?true:false;

for( int i=0; !quit && NoError && i<transV.size();i++){

    Rectangle2D.Double r =
        (Rectangle2D.Double)labelV.elementAt(i);

    //true if mouse point p is within the Rectangle instance
    //
    if(p.getX() >= r.getX() &&
        p.getX() <= r.getX() + r.getWidth() &&
        p.getY() >= r.getY() &&
        p.getY() <= r.getY() + r.getHeight()){

        index4tV = i;
        quit = true;
    }//if
}//for

return index4tV;
}

/**
 * Show the edit transition window for editing the
 * transition.
 */
private void setVisible(JButton b){

    int [] select = {0,0,1};
    LET = new LbaEditTransition(b,inputAlphabet,tapeAlphabet,
        Endmarkers, select);

    LET.setResizable(false);
    LET.setVisible(true);

}

/**
 * Return the coordinate of the state index.
 */
private Point getStatePoint(int index){

    index = findIndex(index);

    Ellipse2D.Double e =
        (Ellipse2D.Double)stateV.elementAt(index);

    return new Point((int)e.getX(),(int)e.getY());

}

/**
 * Returns the state size of the specified state index.
 */
private int getStateSize(int index){

    index = findIndex(index);

```

```

        Ellipse2D.Double e =
            (Ellipse2D.Double)stateV.elementAt(index);

        return (int)e.getWidth();
    } //getStateSize

    /**
     * This method will be invoked by LbaCtrlProcess.class
     * Returns new Stack( Direction, state_index, replaced_char)
     * label format -> sb{"a/b D"}, sb.length = 5.
     * if stack.size == 0, no function defined.
     * if stack.size == 4, there is one transition defined.
     * if stack.size > 4, nondeterministic transition.
     */
    public Object getRange(int stateX, char cScaned){

        Stack stack = new Stack();

        if(startState.intValue() == -1){
            LbaPopup.Warn(0);
        } //if

        stateX = stateX == -1?startState.intValue():stateX;

        for(int i=0;i<fromToV.size();i++){

            int [] index = (int []) fromToV.elementAt(i);

            if(index[0] == stateX){

                StringBuffer sb = new StringBuffer(
                    (String)transV.elementAt(i));

                if(cScaned == sb.charAt(0)){
                    currentState = new Integer(index[1]);

                    stack.push( String.valueOf(i));
                    stack.push(new Character(sb.charAt(2)));
                    stack.push(new Integer(index[1]));
                    stack.push(new Character(sb.charAt(4)));

                } //if input matched
                else{
                    //do nothing.
                } //else no input matched.

            } //if state matched

        } //for

        return stack;

    } //getRange

    /**
     * Returns domain of the transition. For the undo operation.
     */
    public Stack getDomain(int indexTransV){

        Stack domain = new Stack();

```



```

if(indexTransV < 0 || indexTransV >= transV.size()){
    System.out.println("Error@LDA.getDomain");
    System.exit(204);
}

} //if error

int [] index = (int []) fromToV.elementAt(indexTransV);

StringBuffer sb = new StringBuffer(
    (String)transV.elementAt(indexTransV));

domain.push(new Character(sb.charAt(0)));
domain.push(new Integer(index[0]));
domain.push(new Character(sb.charAt(4)));

return domain;

} //getDomain

/**
 * Sets transition label.
 */
private void setTransitionLabel(){

    transV.addElement(sTransiion);
    labelV.addElement(
        new Rectangle2D.Double(
            mouseP.x, mouseP.y,
            LbaData.aLabelSizeX[size], LbaData.aLabelSizeY[size] ));

    int [] aint = {
        ((Integer)indexV.elementAt(iFrontState)).intValue(),
        ((Integer)indexV.elementAt(iToState)).intValue() };
    fromToV.addElement( aint );

    repaint();
} //setTransitionLabel

/**
 * Inner class for action listener of mouse event.
 * The edit transition window Ok button and bEdit
 * radio button.
 */
private class BasicActionListener implements ActionListener{

    public void actionPerformed( ActionEvent e ){
        Object source = e.getSource();

        if( source == b3in1 ){
            // return string in format "a/b d"
            sTransiion = LET.getSelectedItemAt();
            LET.setVisible(false);

            setTransitionLabel();

        } //else if
        else if( source == bEdit ){

            transV.setElementAt(LET.getSelectedItemAt(), editIndex);
            LET.setVisible(false);

        } //else if

```

```

        else{

            System.out.println("Error @ ActionListener.");
        }//else

    }//actionPerformed

} //private class BasicActionListener

} //class LbaDrawArea

/**
 * @author Joji Doi
 * @date 11/14/2000
 * @file LbaCtrlProcess.java
 * @class LbaCtrlProcess
 * @inner class Process
 */

import java.awt.Dimension;
import java.awt.Rectangle;
import java.util.Hashtable;
import java.util.Stack;

import javax.swing.JFrame;

/**
 * Keeps process(es) and manipulate it. STEP, UNDO
 * GO, and PAUSE operation are taken in this class.
 */
public class LbaCtrlProcess{

    /**
     * Stores all existing processes
     * JFrame reference is used as the hashkey.
     */
    private Hashtable processesH;
    private LbaCtrlGraphics canvasLCG;
    private LbaCtrlLabel labelLCL;
    private LbaTreeView treeLTV;
    private LbaCtrlTable tableLCT;

    /**
     * Initialize hashtable for the process pool which stores
     * instances of Process class.
     */
    public LbaCtrlProcess(LbaCtrlGraphics LCG,
        LbaCtrlLabel LCL, LbaTreeView LTV){

        processesH = new Hashtable();
        canvasLCG = LCG;
        labelLCL = LCL;
        treeLTV = LTV;

    } //const.

    /**
     * Undo the transition. If the following condition
     * meets, JOIN operation will be taken place.
     * if( current proces is identical to one in the
     * memory) destroy the current process or one in the

```

```

* memory.
*/
public void undo(Object hashkey, LbaDrawArea LDA){

    Process p = (Process)processesH.get(hashkey);

    int transID = (p.index4transV.length() > 0)
        ?Integer.parseInt( String.valueOf(
            p.index4transV.charAt(p.index4transV.length()-1)))
        :-1;

    if(transID != -1){

        Stack stack = (Stack)LDA.getDomain(transID);

        if(stack.size() != 3){
            System.out.println("error@LCP.undo");
            System.exit(205);
        }

        p.undo(stack);

        p.index4transV = p.index4transV.
            deleteCharAt(p.index4transV.length()-1);

    }

    //error no undo is possible.
    LbaPopup.Error(2);
    //

}

/**
 * Process one transition. If non-deterministic
 * transitions are defined creates new JFrames and
 * adds them to canvasLCG.
 */
public boolean step(Object hashkey, LbaDrawArea LDA){
    final int C_stackSize = 4;
    boolean bHalt = false;

    Process p = (Process)processesH.get(hashkey);

    Stack stack = (Stack)LDA.getRange(p.getState(), p.getChar());
    if(stack.size() == 0){
        LbaPopup.Error(3);
        bHalt = true;
    }

    else if(stack.size() >= C_stackSize &&
        stack.size()%C_stackSize == 0){

        int numOfProcess = stack.size()/C_stackSize;

        if(processesH.size() > LbaData.MaximumNumOfProcesses){
            LbaPopup.Error(4);
        }

    }

```

```

if(numOfProcess > 1){
    LbaPopup.Ask(4, new Integer( numOfProcess ));

} //if

/**
 * Create new processes for nondeterministic
 * transition. Set listener mode off.
 */
Process [] processes = new Process[numOfProcess];
for(int i=0;i<numOfProcess;i++){

    processes[i] = i==0?p:new Process(p);

} //for

LbaTiledWindow LTW = new LbaTiledWindow(numOfProcess,
    ((JFrame)hashkey).getBounds());
Rectangle [] rects = LTW.getRects();
int index = 0;

for(int i=0;i<numOfProcess;i++){

    Stack anUnit = new Stack();
    for(int j=0;j<C_stackSize;j++){
        anUnit.push(stack.pop());

    }

    if( i != 0){

        JFrame newFrame =
            canvasLCG.setCanvas(rects[index++]);

        //In this method Tree node will be added.
        //
        addProcess(hashkey, newFrame, processes[i] );
        newFrame.setVisible(true);

    } //if

    bHalt = processes[i].set(anUnit);

} //for

canvasLCG.setMode(false);

if(numOfProcess>1){
    JFrame temp = (JFrame)hashkey;
    temp.setBounds(rects[index]);
} //if

} //else if
else{
    System.out.println("error@LCP.step.");
    System.exit(201);

} //else
return bHalt;
} //step

/**
 * Removes the process using the accepting object as
 * the hashkey. Also removes the corresponding node

```

```

* form tree view window.
*/
public void deleteProcess(Object hashkey){
    processesH.remove(hashkey);
    treeLTV.deleteNode(hashkey);

} //deleteProcess

/**
* Only once at the very beginning this method will be
* invoked to add process to processesH and tree view.
*/
public void addProcess(Object newKey, String s){
    processesH.put(newKey, new Process(s, labelLCL, canvasLCG));
    treeLTV.addNode(newKey, newKey);

} //addProcess

/**
* Every non-deterministic transition invokes this
* method.
*/
private void addProcess(Object hashkey, Object newKey,
    Process p){

    processesH.put(newKey, p);
    treeLTV.addNode(hashkey, newKey);

} //addProcess

/**
* Copies the reference to table to the private
* instance variable of LbaCtrlTable.
*/
public void addTable(LbaCtrlTable _tableLCT){
    tableLCT = _tableLCT;
} //addTable

/**
* Returns the tape from the specified process.
* If the specified process is null, returns
* empty string.
*/
public String getTapeString(Object hashkey){

    String result = "";
    Process p = (Process)processesH.get(hashkey);
    if(p != null)
        result = p.getString();
    return result;

} //getTapeString

/**
* Returns the tape index from the specified process.
* The process should not be null.
*/
public int getTapeIndex(Object hashkey){
    Process p = (Process)processesH.get(hashkey);

    if(p==null){
        System.out.println("exit LCP.getTapeIndex");
        System.exit(88);
    }
}

```

```

    }
    return p.getIndex();
}

//getTapeIndex

/**
 * The class represents a process. A process contains
 * a state diagram display window and tape string.
 * as well as the current head position and state.
 */
private class Process{

    /**
     * tapeString[tapeIndex]
     */
    private int tapeIndex;
    private StringBuffer tapeString;
    private int currentState;

    /**
     * Keep transition history for undo operation
     * transV contains transition functions.
     */
    private StringBuffer index4transV;

    /**
     * This will be invoked only once at the beginning.
     */
    public Process(String s, LbaCtrlLabel LCL,
        LbaCtrlGraphics LCG){
        tapeString = new StringBuffer(s);
        currentState = -1;

        tapeIndex = 0;

        index4transV = new StringBuffer();
    }

    //const.

    /**
     * This will be invoked for a nondeterministic transition.
     */
    public Process(Process p){
        tapeString = new StringBuffer( p.tapeString.toString() );
        currentState = p.currentState;
        tapeIndex = p.tapeIndex;

        index4transV = new StringBuffer(p.index4transV.toString());
    }

    //const.

    /**
     * get state index
     */
    public int getState(){
        return currentState;
    }

    //getTapeIndex

    /**
     * get char at the read/write header.

```

```

*/
public char getChar(){
    return tapeString.charAt(tapeIndex);
} //getChar

/**
 * Returns the tape string.
 */
public String getString(){
    return tapeString.toString();
} //getString

/**
 * Returns tape index.
 */
public int getIndex(){
    return tapeIndex;
} //getIndex

/**
 * Performs undo operation.
 */
public void undo(Stack s){

    char direction = ((Character)s.pop()).charValue();
    int prevState = ((Integer)s.pop()).intValue();
    char cReplace = ((Character)s.pop()).charValue();

    boolean error = false;

    error = (direction == 'R' && tapeIndex <= 0);
    error = (error || direction == 'L' &&
        tapeIndex >= tapeString.length() - 1);

    if(error){
        LbaPopup.Error(5);
    } //if
    else{

        switch(direction){
            case 'R': --tapeIndex; break;
            case 'L': ++tapeIndex; break;
        } //switch

        currentState = prevState;

        tapeString.setCharAt(tapeIndex, cReplace);

        labelLCL.setText(tapeString.toString(), tapeIndex);
        canvasLCG.getCanvas().currentState =
            new Integer(currentState);

        //Table change
        //
        tableLCT.changeColor(currentState, tapeString.charAt(tapeIndex));

        canvasLCG.getSPscroller().getViewport().repaint();

    } //else
} //undo

```

```

/**
 * Stack( Direction, state_index, replaced_char)
 * Bound check on the tape will be done.
 * Also checks whether current state is the final state.
 */
public boolean set(Stack s){

    boolean bHalt = false;
    String sIndex = (String)s.pop();
    char cReplace = ((Character)s.pop()).charValue();
    int nextState = ((Integer)s.pop()).intValue();
    char direction = ((Character)s.pop()).charValue();

    boolean error = false;

    error = (direction == 'L' && tapeIndex <= 0);
    error = (error || direction == 'R' &&
        tapeIndex >= tapeString.length() -1);

    if(error){
        LbaPopup.Error(10);

        bHalt = true;
    }//if
    else if(index4transV.length() >
        LbaData.MaximumNumOfTransitions){
        LbaPopup.Error(7);

        bHalt = true;
    }//else if
    else{

        index4transV.append(sIndex);
        tapeString.setCharAt(tapeIndex, cReplace);

        currentState = nextState;

        switch(direction){
            case 'L': --tapeIndex; break;
            case 'R': ++tapeIndex; break;
        }//switch

        labelLCL.setText(tapeString.toString(), tapeIndex);
        canvasLCG.getCanvas().currentState =
            new Integer(currentState);

        //Table change
        //
        tableLCT.changeColor(currentState, tapeString.charAt(tapeIndex));

        canvasLCG.getSPscroller().getViewport().repaint();

    }//else

    //Final state check
    //

```



```

        if (canvasLCG.checkFinalState(currentState)) {

            LbaPopup.Infor(0);

            bHalt = true;
        } //if

        return bHalt;

    } //set

    /**
     * Debugging method
     */
    public void print() {
        System.out.print("Process [");
        System.out.println("tapeString " + tapeString);
        System.out.println("currentState " + currentState);
        System.out.println("]");
    } //print
} //class Process

} //class LbaCtrlProcess

/**
 * @author Joji Doi
 * @date 11/14/2000
 * @file LbaData.java
 * @class LbaData
 */

import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumuration;

/**
 * Constants data that are used by any other classes.
 */
public class LbaData {

    /**
     * maximum number of transitions
     */
    public static final int MaximumNumOfTransitions = 10000;

    /**
     * maximum number of processes
     */
    public static final int MaximumNumOfProcesses = 50;

    /**
     * maximum view size
     */
    public static final int MaximumViewSize = 3000;

    /**

```

```

* number of symbols for alphabel domain.
*/
public static final int NumOfLetters = 75;

/**
* maximum number of states to be drawn.
*/
public static final int MaxStates = 100;

/**
* maximum tape length
*/
public static final int MaxLenTape = 1000;

/**
* minimum speed delay is the fastest transition.
*/
public static final int MinimumSpeedDelay = 100;

/**
* initial speed
*/
public static final int InitialSpeedDelay = 800;

/**
* maximum speed delay is the slowest transition.
*/
public static final int MaximumSpeedDelay = 1000;

/**
* URL for online program documentation.
*/
public static final String URL_DOC =
    "http://www.cs.okstate.edu/~doi/lba/doc/index.html";

/**
* URL for online evaluation form
*/
public static final String URL_EVALUATION =
    "http://139.78.81.243/cgi-bin/evaluation.asp";

/**
* Shared environment data path.
*/
public static final String PATH_UNIX =
    "/d/doi/lba/images/";

/**
* Shared environment data path.
*/
public static final String PATH_WINDOWS =
    "images/";
    //"d:/Programming/SwingBasedTool/images/";

/**
* Shared environment data path.
*/
public static final String FILE_PATH_UNIX =
    "/d/doi/lba/files/";

/**
* Shared environment data path.

```

```

*/
public static final String FILE_PATH_WINDOWS =
    "files/";
    //"d:/Programming/SwingBasedTool/files/";

/**
 * State sizes for zooming operation.
 */
public static final int [] aStateSize
    = {14,17,20,23,27,32,38,44,52};

/**
 * The state index. It can be modified to be user
 * specified index instead of constant.
 */
public static final String StateIndex = "q";

/**
 * aFontSize is the same as the arrowhead size
 * for zooming operation.
 */
public static final int [] aFontSize
    = { 8, 9,10,11,12,13,14,15,16};

/**
 * Transition label sizes width for zooming operation.
 */
public static final int [] aLabelSizeX = {
    29,32,35,45,50,55,60,65,70
};

/**
 * Transition label sizes height for zooming operation.
 */
public static final int [] aLabelSizeY = {
    9,10,10,14,16,20,20,26,29
};

/**
 * Creates an array of characters for the alphabet.
 * Number of letters must be matches with NumOfLetters.
 */
public static void alphabet( char [] aLetters ){
    int i = 0;
    for(char c = 'A'; c<='Z'; i++, c++){
        aLetters[i] = c;
    }
    for(char c = 'a'; c<='z'; i++, c++){
        aLetters[i] = c;
    }
    for(char c = '0'; c<='9'; i++, c++){
        aLetters[i] = c;
    }
    for
    //13
    aLetters[i++] = '+';
    aLetters[i++] = '-';
    aLetters[i++] = '*';
    aLetters[i++] = '/';
    aLetters[i++] = '%';

```

```

aLetters[i++] = '!';
aLetters[i++] = '^';
aLetters[i++] = '=';
aLetters[i++] = '?';
aLetters[i++] = '&';
aLetters[i++] = '|';
aLetters[i++] = '{';
aLetters[i++] = '}';

} //alphabet

/**
 * Returns the array of string object which
 * contains input alphabet and endmarkers.
 */
public static String [] alphabet( Hashtable h,
    Character [] EndMarkers ){

    Enumeration e = h.elements();

    String result [] = new String[h.size() + 2];

    for(int i=0;i<result.length;i++){
        switch(i){
            case 0:
            case 1: result[i] = EndMarkers[i].toString(); break;
            default: result[i] = (String)e.nextElement();

        } //switch

    } //for
    return result;

} //alphabet

/**
 * End-markers domain. Must be distinguished from
 * alphabet domain.
 */
public static final char [] endMarkers = {
    '$','&', '@', '<', '>', '#', '{', '}', '[', ']'
};

/**
 * GUI menu, menu item, button, icon, and list names.
 */
public static final String [] Text = {
    // 0 - 9
    "Create a state", "Create a transition",
    "Delete a state", "Delete a transition",
    "Move a state / transition",
    "Move selected area", "Select the start state",
    "Select the final state", "Status", "New",
    // 10 - 19
    "Open File", "Save As...", "Exit", "File",
    "Set Input Alphabet", "Set End Markers",
    "Set Tape Alphabet", "Create/Edit Tape String",
    "Configure", "Transition Table view option",
    // 20 - 29
    "Tree view option", "Tape view option", "Speed Ctrl",
    "View", "Documentation", "Run time guide",
    "About Linear-Bounded Automaton...", "Help",
    "EDIT", "RUN",
    // 30 - 39

```

```

        "STEP", "UNDO", "GO", "PAUSE", "ZOOM IN", "ZOOM OUT",
        "Print", "Edit a transition", "Open Tape", "Save Tape As..."
    };
} //class LbaData

/**
 * @author Joji Doi
 * @date 11/14/2000
 * @file LbaCtrlZoom.java
 * @class LbaCtrlZoom
 */

import java.awt.Dimension;
import java.awt.Point;
import java.awt.geom.Point2D;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import java.util.Vector;

/**
 * Zoom operations. Contains two static methods.
 * One method perform zoom in operation. The other
 * does zoom out operation.
 */
public class LbaCtrlZoom{

    /**
     * Zoom in operation modifies the location of state and
     * transition label object.
     * NewLocationX = OriginalStateX + DistanceWidth / 2,
     * NewLocationY = OriginalStateY + DistanceHeight / 2
     */
    public static void IN(Vector stateV, Vector labelV,
        Point2D.Double centerFrame, int zoomLevel){

        boolean zoomable;

        //index check
        //
        zoomable =
            ( zoomLevel >= 0 && zoomLevel < LbaData.aStateSize.length )
            ? true : false;

        //compute the distance between two points. for rect.
        //and ellipse.
        //

        //Modify values on state and label objects
        //

        for(int i=0; zoomable && i<stateV.size(); i++){

            //State
            //
            Ellipse2D.Double e = (Ellipse2D.Double)stateV.elementAt(i);
            Point2D.Double elliP =
                new Point2D.Double(e.getX(), e.getY());

            Dimension distance =
                new Dimension( (int)(elliP.getX() - centerFrame.getX()),
                    (int)(elliP.getY() - centerFrame.getY()) );

```

```

//XXX This state is on the left side of the window.
//XXX Or it can be on the right side of the window.
//XXX This state is on a top of the window.
//XXX This state is on a bottom side of the window.

elliP.setLocation( elliP.getX() + distance.getWidth()/2,
    elliP.getY() + distance.getHeight()/2);

e.setFrame(elliP.getX(), elliP.getY(),
    LbaData.aStateSize[ zoomLevel ],
    LbaData.aStateSize[ zoomLevel ]);

} //for

for(int i=0; zoomable && i<labelV.size();i++){

    //Label box
    //
    Rectangle2D.Double r =
        (Rectangle2D.Double)labelV.elementAt(i);
    Point2D.Double rectP =
        new Point2D.Double(r.getX(), r.getY());
    Dimension distanceL =
        new Dimension( (int)(rectP.getX() - centerFrame.getX()),
            (int)(rectP.getY() - centerFrame.getY()));
    rectP.setLocation( rectP.getX() + distanceL.getWidth()/2,
        rectP.getY() + distanceL.getHeight()/2);
    r.setFrame(rectP.getX(), rectP.getY(),
        LbaData.aLabelSizeX[ zoomLevel ],
        LbaData.aLabelSizeY[ zoomLevel ]);

} //for
} //zoomIN

/**
 * Zoom out operation modifies the location of state and
 * transition label object.
 * NewLocationX = OriginalStateX - DistanceWidth / 3,
 * NewLocationY = OriginalStateY - DistanceHeight / 3
 */
public static void OUT(Vector stateV, Vector labelV,
    Point2D.Double centerFrame, int zoomLevel){

    boolean zoomable;

    //index check
    //
    zoomable =
        ( zoomLevel >= 0 && zoomLevel < LbaData.aStateSize.length )
        ?true:false;

    for(int i=0; zoomable && i<stateV.size();i++){

        Ellipse2D.Double e = (Ellipse2D.Double)stateV.elementAt(i);
        Point2D.Double elliP =
            new Point2D.Double(e.getX(), e.getY());

        Dimension distance =
            new Dimension( (int)(elliP.getX() - centerFrame.getX()),
                (int)(elliP.getY() - centerFrame.getY()) );

```

```

//XXX This state is on the left side of the window.
//XXX Or it can be on the right side of the window.
//XXX This state is on a top of the window.
//XXX This state is on a bottom side of the window.

elliP.setLocation( elliP.getX() - distance.getWidth()/3,
    elliP.getY() - distance.getHeight()/3);

e.setFrame(elliP.getX(), elliP.getY(),
    LbaData.aStateSize[ zoomLevel ],
    LbaData.aStateSize[ zoomLevel ]);

} //for
for(int i=0; zoomable && i<labelV.size();i++){
    //Label box
    //
    Rectangle2D.Double r =
        (Rectangle2D.Double)labelV.elementAt(i);
    Point2D.Double rectP =
        new Point2D.Double(r.getX(), r.getY());

    Dimension distanceL =
        new Dimension( (int)(rectP.getX() - centerFrame.getX()),
            (int)(rectP.getY() - centerFrame.getY()));
    rectP.setLocation( rectP.getX() - distanceL.getWidth()/3,
        rectP.getY() - distanceL.getHeight()/3);
    r.setFrame(rectP.getX(), rectP.getY(),
        LbaData.aLabelSizeX[ zoomLevel ],
        LbaData.aLabelSizeY[ zoomLevel ]);

} //for
} //zoomOUT

} //class LbaCtrlZoom

/**
 * @author Nobuo Tamemasa (http://www2.gol.com/users/tame/).
 * @date 11/14/2000 (original 11/22/1998)
 * @file CellAttributiveRenderer.java
 * @class CellAttributiveRenderer
 */

import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.border.*;

/**
 * Manipulate the table cell of colors and values.
 */
public class CellAttributiveRenderer extends JLabel
implements TableCellRenderer{

    protected static Border noFocusBorder;

    /**
     * Constructor initialize the border.
     */
    public CellAttributiveRenderer(){
        noFocusBorder = new EmptyBorder(1, 2, 1, 2);
        setOpaque(true);
        setBorder(noFocusBorder);
    } //const.

```

```

/**
 * Attribute of a cell to be set based on the cell
 * condition. (i.e., focused, selected, and so on ).
 */
public Component getTableCellRendererComponent(
    JTable table, Object value, boolean isSelected,
    boolean hasFocus, int row, int column){

    Color foreground = null;
    Color background = null;
    Font font = null;
    TableModel model = table.getModel();

    if(model instanceof CellAttributiveTableModel){
        CellAttribute cellAtt =
            ((CellAttributiveTableModel)model).getCellAttribute();

        if(cellAtt instanceof CellColor){
            foreground =
                ((CellColor)cellAtt).getForeground(row,column);
            background =
                ((CellColor)cellAtt).getBackground(row,column);
        }//if

        if(cellAtt instanceof CellFont){
            font = ((CellFont)cellAtt).getFont(row,column);
        }//if
    }//if
    if(isSelected){
        setForeground((foreground != null) ? foreground
            : table.getSelectionForeground());
        setBackground(table.getSelectionBackground());
    }//if
    else{
        setForeground((foreground != null) ? foreground
            : table.getForeground());
        setBackground((background != null) ? background
            : table.getBackground());
    }//else

    setFont((font != null) ? font : table.getFont());

    if(hasFocus){
        setBorder( UIManager.getBorder(
            "Table.focusCellHighlightBorder" ) );

        if(table.isCellEditable(row, column)) {

            setForeground((foreground != null)
                ? foreground
                : UIManager.getColor("Table.focusCellForeground" ) );

            setBackground( UIManager.getColor(
                "Table.focusCellBackground" ) );
        }//if
    }//if
    else {
        setBorder(noFocusBorder);
    }//else
    setValue(value);
    return this;
} //getTableCellRendererComponent

/**
 * Set string value to the cell.

```



```

    * If the string value is null, sets empty string.
    */
    protected void setValue(Object value) {
        setText((value == null) ? "" : value.toString());
    } // setValue
} // CellAttributiveRenderer

/**
 * @author Joji Doi
 * @date 9/2/2000
 * @file QuadraticForm.java
 * @class QuadraticForm
 */

import java.awt.Point;
import java.awt.geom.Point2D;
/**
 * An instance of this class will calculates the
 * intersection of a state and an arrow point
 * i.e.,
 * There are two states. Name them S_1 and S_2.
 * Assumption is that these states are circle
 * and their radiuses are the identical to
 * each other.
 * Their center positions are known as point_1
 * and point_2, respectively.
 * There will be two intersections if a
 * straight line is drawn between point_1 and
 * point_2.
 * Returns the intersection points which needs
 * an arrowhead.
 */
public class QuadraticForm{

    private boolean normal = true;

    private Point2D.Double center, center2;
    private Point2D.Double xy;

    /**
     * This constructor is used to initialize the data member
     * for floating point (double) coordinates.
     */
    public QuadraticForm(
        Point _p1, Point2D.Double p2, double radius) {

        Point2D.Double p1 = new Point2D.Double(_p1.x, _p1.y);

        C_QuadraticForm(p1, p2, radius);

    } // constructor

    /**
     * This constructor is used to initialize the data member
     * for integer coordinates.
     */
    public QuadraticForm(Point _p1, Point _p2, double radius) {

        Point2D.Double p1 = new Point2D.Double(_p1.x, _p1.y);
        Point2D.Double p2 = new Point2D.Double(_p2.x, _p2.y);
    }

```

```

        C_QuadraticForm(p1,p2,radius);

    }//constructor

    /**
     * This constructor is used to initialize the data member
     * for label intersection with a curved line.
     */
    public QuadraticForm(Point _p1,Point _p2,
        double radius,boolean b){
        normal = !normal;

        Point2D.Double p1 = new Point2D.Double(_p1.x,_p1.y);
        Point2D.Double p2 = new Point2D.Double(_p2.x,_p2.y);

        C_QuadraticForm(p1,p2,radius);

    }//constructor

    /**
     * Returns the intersection coordinate for given
     * data value.
     */
    public Point getIntersection(){
        return new Point((int)xy.x,(int)xy.y);
    }//getIntersection

    /**
     * Caluculates the coordinate using quadratic formula.
     * Need to test which of states needs arrowhead.
     * check line vector, add one more element for int.
     * This int value tells program:
     * -1 no arrowhead required.
     * state number which state needs indegree arrowhead.
     * then search the state, and get center point of that
     * state. and send the Point value to the next
     * method. =====> "center" <--- point of the state.
     */
    private void C_QuadraticForm(
        Point2D.Double p1,Point2D.Double p2,double radius){

        //To avoid x1=x2. add x1+1;
        //if x1=x2, divisor will be 0 and x is NAN.
        if(p1.x == p2.x) p1.x+=0.1;

        center = p1;
        center2 = p2;

        double slope = getSlope(center,center2);
        double b = center.y-(slope*center.x);
        double b2 = center2.y-(slope*center2.x);

        double x = getIntersection(radius,b,slope,center);

        xy = getPoint(x,slope,b);

    }//C_QuadraticForm

    /**
     * Computes y coordinate and returns new coordinate.
     */

```

```

private Point2D.Double getPoint(
    double x, double a, double b){
    Point2D.Double temp;
    double y = a*x+b;
    temp = new Point2D.Double(x,y);
    return temp;
} //getPoint

/**
 * Computes and returns slope of the accepting values of
 * dcoordinates.
 */
private double getSlope(Point2D.Double center,
    Point2D.Double center2){
    return (center.y-center2.y)/(center.x-center2.x);
} //getSlope

/**
 * This method need to decide which of two intersection points
 * is appropriate point to have arrowhead.
 * returned X value must be appropriate value of p and n.
 */
private double getIntersection(double radius, double b, double slope,
    Point2D.Double o){

    double a = slope;
    double j = o.x;
    double k = o.y;
    double r = radius;
    double alpha = 1+(a*a);
    double beta = (2.0*a*b)-(2.0*j)-(2.0*a*k);
    double gama = (k*k)-(r*r)+(j*j)+(b*b)-(2*b*k);

    double p = whatIsPositiveX(alpha,beta,gama);
    double n = whatIsNegativeX(alpha,beta,gama);

    double result = 0;

    if(center.x < center2.x){
        result = normal?p:n;
    } //if
    else{
        result = normal?n:p;
    } //else

    return result;
} //getIntersection

/**
 * Returns positive value of x coordinate.
 */
private double whatIsPositiveX(double a, double b, double c){
    double x = (-b+Math.sqrt((b*b)-(4.0*a*c)))/(double)(2.0*a);
    return x;
} //whatIsPositiveX

/**
 * Returns negative value of x coordinate.
 */
private double whatIsNegativeX(double a, double b, double c){
    double x = (-b-Math.sqrt((b*b)-(4.0*a*c)))/(double)(2.0*a);
    return x;
} //whatIsNegativeX
} //class QuadraticForm

```

VITA 8

Joji Doi

Candidate for the Degree of

Master of Science

Thesis: A GRAPHICAL SIMULATOR FOR TWO-WAY LINEAR-BOUNDED
AUTOMATA (LBA)

Major Field: Computer Science

Biographical:

Personal Data: Born in Tokyo, Japan, May 29, 1974, son of Mr. and Mrs.
Junji and Etsuko.

Education: Graduated from Ogoose High School, Saitama, Japan, March 1993;
received Associate of Arts in Management from Dohto University Junior
College Division, Hokkaido, Japan, March 1995; received Bachelor of
Science in International Business from Baker University, Baldwin City,
Kansas, May 1997. Completed the requirements for the Master of Science
in Computer Science at the Computer Science Department at Oklahoma
State University, May 2001.

